Just-in-time deception to detect credential-stuffing bots

Abhishek Singh, Manish Sardiwal, Ramesh Mani

www.prismosystems.com

Introduction Manish Sardiwal

- Staff Security Research Engineer at Prismo Systems
- Previously : Senior Research Scientist at FireEye , Quick Heal
- Led analysis of 0 day, known Vulnerabilities, development of anti-exploitation techniques, algorithms to detect lateral movement.

Introduction Abhishek Singh

- Chief Researcher at Prismo Systems
- Previously : Spearheaded Threat R&D at FireEye, Microsoft, Acalvio
- 30 Patents (Issued, Pending) Detection Algorithms, Analytics & Architecture of Detection Technologies.
- 2019 Reboot Leadership Award (Innovators Category): SC Media
- Nominee for Prestigious Virus Bulletin’s 2018 Péter Szőr Award.

Introduction Ramesh Mani

- Distinguished Engineer at Prismo Systems
- Previously: Spearheaded R&D of APM agents at CA Technologies, Wily
- 12+ Patents issued.

Agenda

- Introduction and classification of Credential Stuffing Bots
- Dynamic deception algorithm to detect Bots
- Implementation and Architecture
- Demo

Credential Stuffing Bots

This report is like a mini time capsule – the original report remains intact. We dive deep into the data from January 1, 2018, to December 31, 2019, and keep the original analysis of how credential stuffing has impacted media companies. When we look just at the media sector during this period, we can see that Akamai recorded 17 billion credential stuffing attacks during that 24-month period.

On March 26, 2020, a video media service in Europe experienced a strong spike in attacks, reaching 348,050,675 malicious login attempts in 24 hours, accounting for 96% of all malicious login attempts against the video media sub-vertical during the same period. This service provider was one of several that were targeted, as credential stuffing attacks reached peaks in the hundreds of millions daily during Europe's lockdown.

by: CNN Wires

Posted: Aug 17, 2020 / 01:27 PM CDT / Updated: Aug 17, 2020 / 01:27 PM CDT

The Canadian government said it was forced to shut down most of its online portals on the weekend after a sustained cyberattack over the last several days.

At one point over the weekend, Canadian officials disclosed they detected as many as 300,000 attempted attacks to access accounts on at least 24 government systems.

Why Dynamic Deception for Bot Detection?

Dynamic Deception -- Injecting Breadcrumbs & nonce at runtime has the following benefits

- Bots get detected on their very first malicious login attempt
    - -- Malicious Traffic will be reduced instantly.
- Precise Alert , validated against the knows family's of Credential Stuffing Bots.

Classification of Credential Stuffing Bot

Credential stuffing bots can be divided in two major categories based upon the manner in which stolen credentials are entered:

- Loads and scrapes the HTML login page and submit the login form
- Uses login APIs to submit the login request

First Category of the Bot: Loads the form

- Loads the website login page
- Scrapes the login form tag, username and password input fields by using attributes like id, name or action of form tag
- Set input fields with stolen credentials and submit the form
- Common libraries used Selenium, Google Puppeteer, Microsoft Playwright, Phantomjs Headless Browser, Python Mechanicalsoup etc
- Examples are Cr3dOv3r, OpenBullet 1.2, Javascript stuffer, Facebook-BruteForce, GUI-form-brut, Credmap, Cloudflare-scrape etc

Example Code : First Category of Bot

```
import mechanicalsoup as ms
browser = ms.StatefulBrowser()
try:
    browser.open(url)
    browser.select_form(form)
    browser[e_form] = email
    browser[p_form] = pwd
    browser.submit_selected()
except ms.utils.LinkNotFoundError:
    error("[{:10s}] Something wrong with the website maybe it's blocked!".format(name))
    return

if is_there_captcha(browser.get_current_page().text):
    error("[{:10s}] Found captcha after submitting login page!".format(name))
    return
```

Second Category of Bot : Uses login API directly

- These bots make GET or POST requests directly to the APIs responsible for login
- These bots do not load and scrape the HTML login page.
- Examples of this category bots are Apex, Snipr, Black Bullet, Storm, Private Keeper, Sentry MBA, Woxy, thc-hydra, Brute Forcing etc

Example Code : Second Category of Bots

Python request library for POST request.

```
# iterate through the list of usernames and give some randomness to the credentials
for username in usernames:
    username = username + str(random.randint(1, 12)) + random.choice(domains)
    # create a pseudo-random alphanumeric string with special chars between 12 and 2
    password = ''.join(
        [random.choice(string.ascii_letters + string.digits + '!@#$*()-+') for x in
    try:
        r = requests.post(url, allow_redirects=False, headers=headers, verify=False,
                data={username_header: username, password_header: password})
        if r.status_code is not 200:
            # just in case something is allowed but doesnt accept the stuffed cred
            raise requests.RequestException
        else:
            print('Sending {!s} with a password of {!s}'.format(username, password))
    except requests.RequestException as error:
        print('Received HTTP ' + str(r.status_code))
        print(error)
        exit(1)
```

Detection using Deception

- Credential stuffing bots parse the login html page using libraries like Selenium, Google Puppeteer, Phantomjs, Python Mechanicalsoup.
- Locate the login form by using DOM elements using attributes like id of element, name of element, action of the login form etc.
- Crafted & injected honey form tag having same attributes as real form tag in the login page which will get accessed prior to real login form by the bots for credential stuffing.
- Login request received from honey form tags gets validated by the algorithm for detection of credential stuffing Bot.

Authentication Validation Algorithm

Each time a login page is served

- Obfuscated Honey Form tags having actual username, password data fields, action URL is inserted in the specific places in the outgoing login pages.
- Random number `{H1...}` is inserted in the Honey `<form>` tags and `R1` in the Real `<form>` tags.

Keys stored by the Algorithm

- Symmetric Keys = $SK_{HoneyForm}$, $SK_{RealForm}$
- Hashing Keys = $SHA\text{-}3_{HoneyForm}$, $SHA\text{-}3_{RealForm}$

Login Page Having Random ID

```
<html>
▼<head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
    <title>
    Website Login
    </title>
  </head>
▼<body bgcolor="lightgray" onload="addform();">
  ▼<form method="POST" id="loginform" action="login" style="position: absolute; height: 0px;
  width: 0px; opacity: 0;">
      <input type="text" name="UserId" id="UserID" style="position: absolute; height: 0px; width:
      0px; opacity: 0;">
      <input type="text" name="Password" id="Password" style="position: absolute; height: 0px;
      width: 0px; opacity: 0;">
      <input type="hidden" name="RandomID" value=
      "bcb5385a37ad83847c41cbfb77107147d1e99e4c3a9fc7be81af9c16d7b893c49a9469ae2792be48225ca272977b
      48ffeafb3369ae255ba49e71afca80c61bddd9b458ddc4b17dbd00796f8115ff5e795cd55fec3b69e6098a31ffea9
      e1aa9a325c82ce8c53c5144a9389d51123bac6d4d77f13acd7f99eadef415a8dff9ddd6f8af86e27abcdfbddfd039
      b37e3e24381418f54f4b2376bf3b2edadf4c4160337a9cb6cc85e5000feb9cc83cda58b878d00790c9d4">
      <input type="submit" value="submit" style="position: absolute; height: 0px; width: 0px;
      opacity: 0;">
    </form>
    <h1>Login</h1>
  ▶<div style="background-color:cyan">…</div>
```

Authentication Validation Algorithm

Computation of the Random Number:

Random Number {H1…} = $(SK_{HoneyForm}(current\_time, SHA\text{-}3_{HoneyForm}(current\_time)))$

Random Number R1 = $(SK_{RealForm}(current\_time, SHA\text{-}3_{RealForm}(current\_time)))$

When the submit button is clicked in the <form> tag, the random number value will send back along with the entered username and password

- {H1...} will be sent from the the Honey <form> tag.
- R1 will be sent from the Real <form> tag.

Authentication Validation Algorithm.. Detection

- First category of Bot: Scraps the form, so it will be accessing honey form tag. `{H1...}` will be present with the username and password in the incoming `POST` response.

  - Parse the incoming login and password request and extract Random value $(RV_{incoming})$

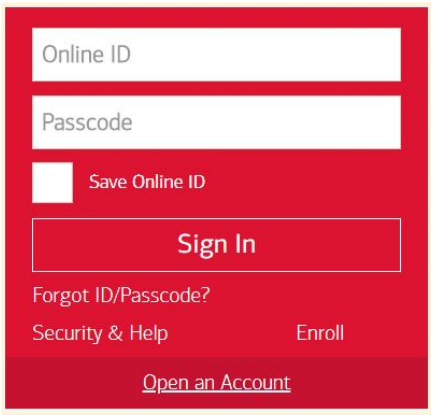
  - Decrypt $SK_{HoneyForm}(RV) = \{time_{incoming\_request}, SHA\text{-}3(time_{incoming\_request})_{incoming}\}$

  - If $\{SHA\text{-}3_{HoneyForm}(time_{incoming\_request}) = SHA\text{-}3(time_{incoming\_request})_{incoming}\}$ = BoT

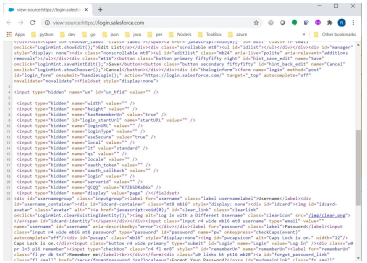
- For the second category of Bot: Uses Rest API's to directly send Username and password `R1` will be missing in the `POST` response

Architecture

HTTP GET for the login page



Instrumented login page



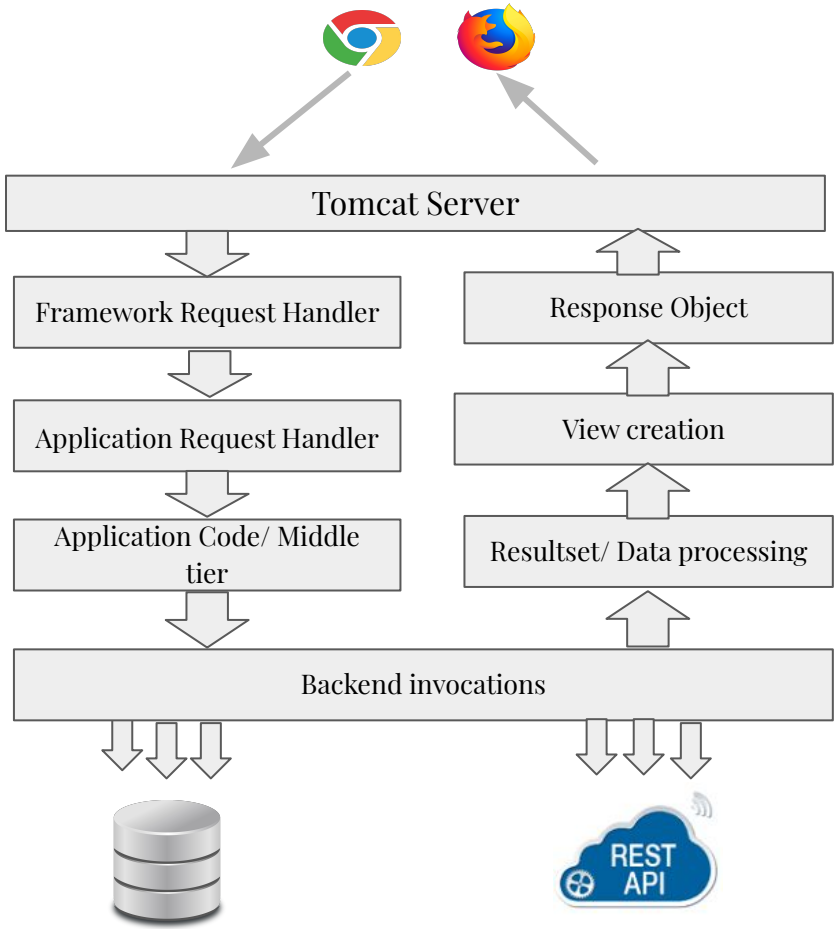
Login Page HTML -- Original

```
<!DOCTYPE HTML>
<html lang="en" xmlns:og="http://ogp.me/ns#">
    <head>
    <meta http-equiv="x-ua-compatible" content="IE=Edge"/>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1, user-scalable=no"/>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
    <link rel="stylesheet" href="/etc.clientlibs/dcu/clientlibs/clientlib-dependencies.min.css" type="text/css">
    <link rel="stylesheet" href="/etc.clientlibs/dcu/clientlibs/clientlib-site-layout.min.css" type="text/css">
    <title>ABC | Personal | Business Banking</title>
 </head>
  <body>

      <form action="https://www.abc.com/app/initialLogin" class="form-signin p-0" autocomplete="off" id="loginForm" method="POST">
     <input type="text" id="userid" class="form-control rounded-0" name="userid" />
     <input type="password" id="password" class="form-control rounded-0"name="password"/>
     <button class="btn btn-tilt text-uppercase rounded-0" role="button" aria-label="Clickable LOGIN button">LOGIN</button>
      </form>

 </body>
 </html>
```

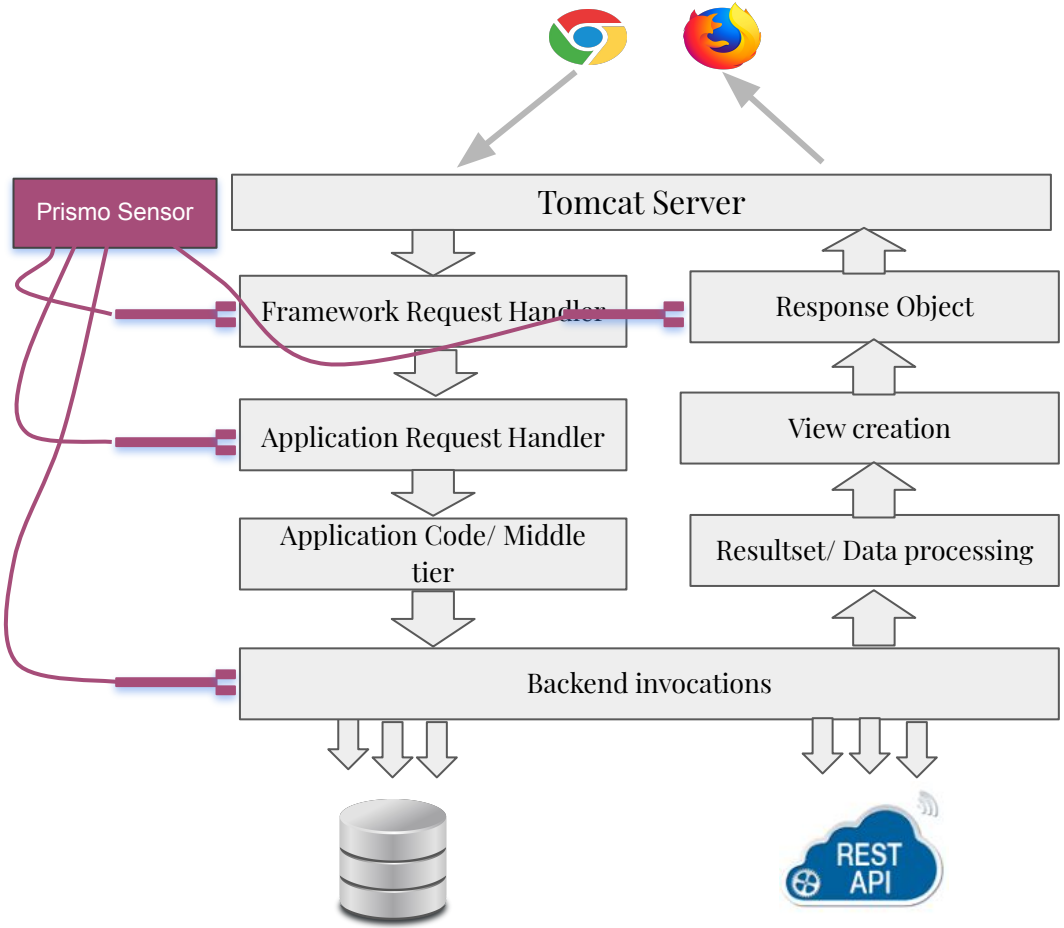
Login Page HTML -- Instrumented

```
<!DOCTYPE HTML>
<html lang="en" xmlns:og="http://ogp.me/ns#">
    <head>
    <meta http-equiv="x-ua-compatible" content="IE=Edge"/>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1, user-scalable=no"/>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
    <link rel="stylesheet" href="/etc.clientlibs/dcu/clientlibs/clientlib-dependencies.min.css" type="text/css">
    <link rel="stylesheet" href="/etc.clientlibs/dcu/clientlibs/clientlib-site-layout.min.css" type="text/css">
    <title>ABC | Personal | Business Banking</title>
     <form action="https://www.abc.com/app/initialLogin" class="form-signin p-0" autocomplete="off" id="loginForm" method="POST">
      <input type="text" id="userid" class="form-control rounded-0" name="userid" />
      <input type="password" id="password" class="form-control rounded-0"name="password"/>
      <input type="hidden" id="logintoken" name="logintoken" value="EncryptedValue-Invalid" />
     <button class="btn btn-tilt text-uppercase rounded-0" role="button" aria-label="Clickable LOGIN button">LOGIN</button>
      </form>
 </head>
  <body>
     <form action="https://www.abc.com/app/initialLogin" class="form-signin p-0" autocomplete="off" id="loginForm" method="POST">
      <input type="text" id="userid" class="form-control rounded-0" name="userid" />
      <input type="password" id="password" class="form-control rounded-0"name="password"/>
      <input type="hidden" id="logintoken" name="logintoken" value="EncryptedValue-Valid" />
     <button class="btn btn-tilt text-uppercase rounded-0" role="button" aria-label="Clickable LOGIN button">LOGIN</button>
      </form>
 </body>
 </html>
```

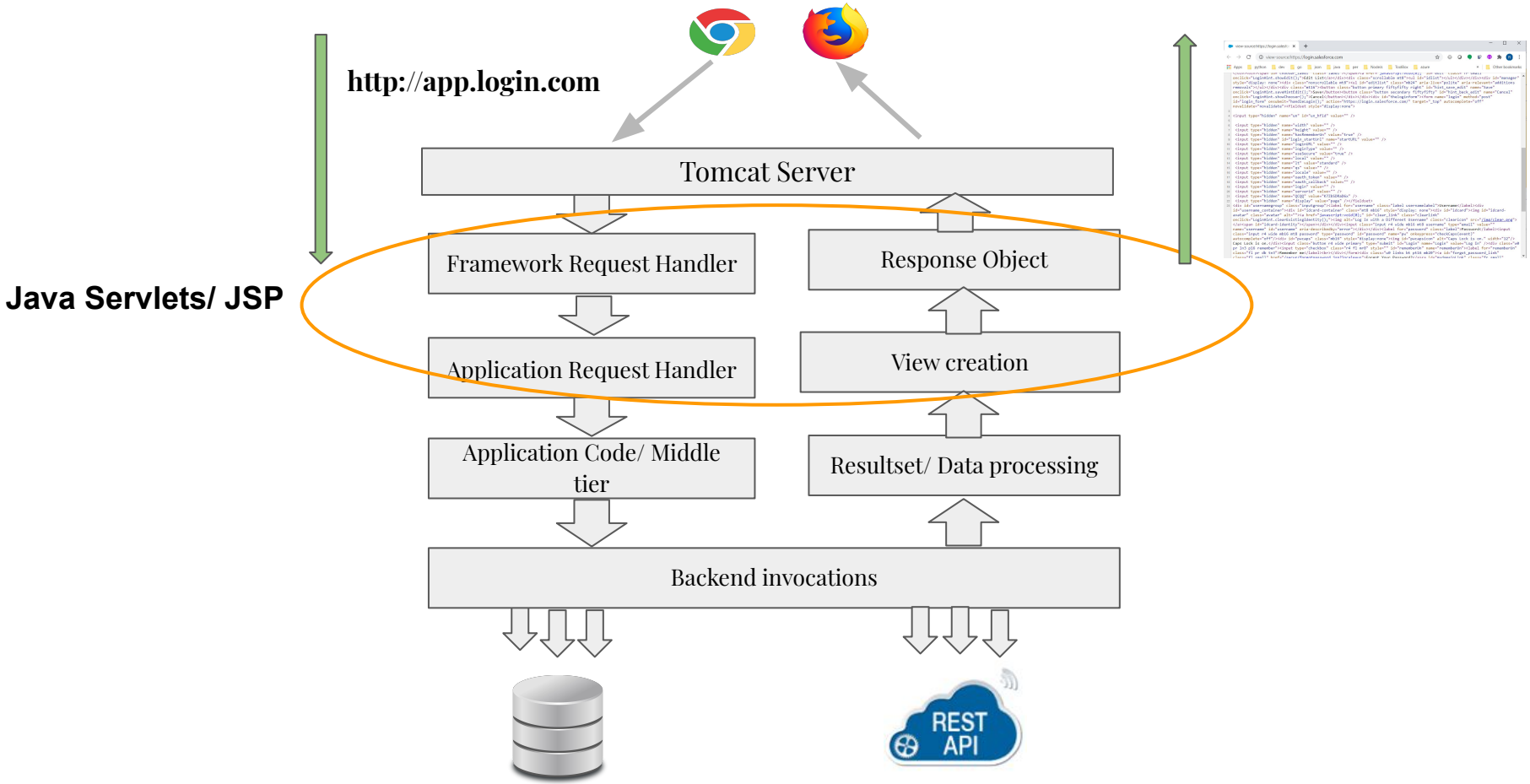
Architecture



Architecture

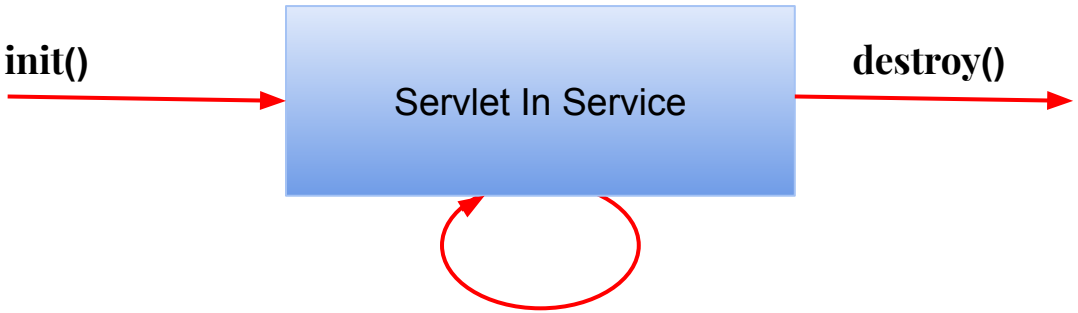


Architecture



Architecture

- Instrument servlets
- Lifecycle of servlets



HTTP:
doGet(), doPost(), doHead(),
doOptions(), etc

Architecture

- Instrument servlets
- Lifecycle of servlets
- Response Wrapper

Interface : `javax.servlet.Servlet`

Method : `void service(ServletRequest req, ServletResponse res)`

Class : `javax.servlet.http.HttpServletResponseWrapper`

Method: `public` **`ServletOutputStream`** `getOutputStream() throws IOException`

⇒ `Extend ServletOutputStream, wrap original`

Method: `public` **`PrintWriter`** `getWriter() throws IOException`

⇒ `Extend PrintWriter, wrap original`

Architecture

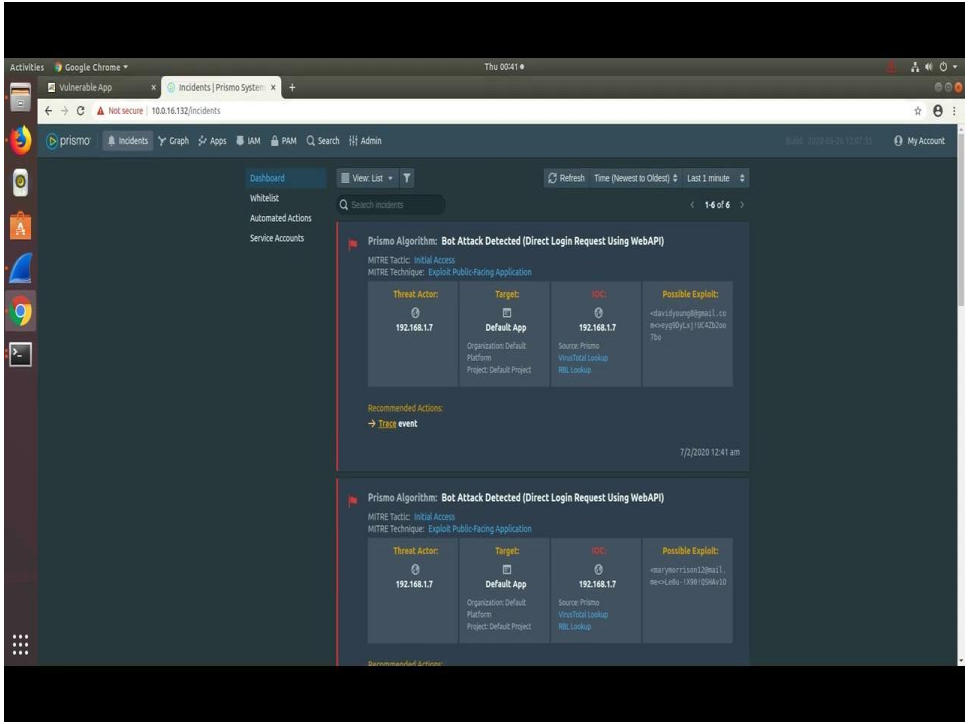
- PrintWriter / OutputStream wrappers
  - Buffer response till </head> tag is seen
  - Insert honey form with encrypted field
  - Continue buffering till app login <form> tag is seen
  - Insert hidden encrypted field in login form

```
<body>
   <form action="https://www.abc.com/app/initialLogin" class="form-signin p-0" autocomplete="off" id="loginForm" method="POST">
    <input type="text" id="userid" class="form-control rounded-0" name="userid" />
    <input type="password" id="password" class="form-control rounded-0"name="password"/>
    <input type="hidden" id="logintoken" name="logintoken" value="EncryptedValue-Valid" />
   <button class="btn btn-tilt text-uppercase rounded-0" role="button" aria-label="Clickable LOGIN button">LOGIN</button>
    </form>
```

Architecture

- Login POST request with credentials
  - Get form field values (Instrumented Servlet)
  - Remove hidden encrypted field from form
  - Let application process request
  - Send data to server (encrypted field and username)

Demo


Activities
Google Chrome
Thu 00:41
Vulnerable App
Incidents | Prismo Syster
Not secure | 10.0.16.132/incidents
prismo
Incidents
Graph
Apps
IAM
PAM
Search
Admin
My Account
Dashboard
Whitelist
Automated Actions
Service Accounts
View: List
Refresh
Time (Newest to Oldest)
Last 1 minute
Search incidents
1-6 of 6
Prismo Algorithm: Bot Attack Detected (Direct Login Request Using WebAPI)
MITRE Tactic: Initial Access
MITRE Technique: Exploit Public-Facing Application
Threat Actor:
192.168.1.7
Target:
Default App
Organization: Default Platform
Project: Default Project
IOC:
192.168.1.7
Source: Prismo
VirusTotal Lookup
RBL Lookup
Possible Exploit:
Recommended Actions:
→ Trace event
7/2/2020 12:41 am
Prismo Algorithm: Bot Attack Detected (Direct Login Request Using WebAPI)
MITRE Tactic: Initial Access
MITRE Technique: Exploit Public-Facing Application
Threat Actor:
192.168.1.7
Target:
Default App
Organization: Default Platform
Project: Default Project
IOC:
192.168.1.7
Source: Prismo
VirusTotal Lookup
RBL Lookup
Possible Exploit:

Thank You

Contact : {asingh,msardiwal, rmani}@prismosystems.com

www.prismosystems.com