

Strategic and Technical Blog >> Research

Unauthenticated Command Injection In Progress Kemp LoadMaster

David Yesland

Introduction

While researching the Progress Kemp LoadMaster load balancer we discovered an unauthenticated command injection in the administrator web interface of the appliance. This allowed full compromise of the LoadMaster if you could reach the administrator web user interface (WUI). The vulnerability was assigned CVE-2024-1212 and has since been patched.

Affected Product Summary

Vendor: Progress Software

Product: Kemp LoadMaster

Confirmed affected version: 7.2.59.0.22007

Patched version: 7.2.59.2.22338

Vendor Advisory:

[https://support.kemptechnologies.com/hc/en-us/articles/23878931058445-](https://support.kemptechnologies.com/hc/en-us/articles/23878931058445-LoadMaster-Security-Vulnerability-CVE-2024-1212)

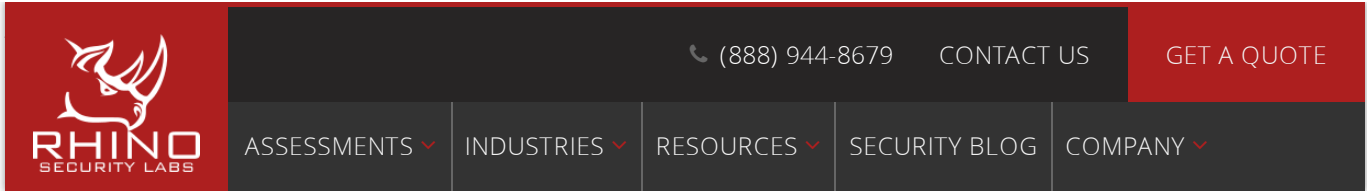
[LoadMaster-Security-Vulnerability-CVE-2024-1212](https://support.kemptechnologies.com/hc/en-us/articles/23878931058445-LoadMaster-Security-Vulnerability-CVE-2024-1212)

What is LoadMaster

LoadMaster is a load balancer and application delivery controller. It ensures high availability and reliable performance for web and application servers by distributing traffic efficiently. LoadMaster can be deployed in various environments to suit different business needs, including options for hardware, virtual machines, and cloud platforms (AWS and Azure).

During this research we used the [virtual machine image](#) to test the LoadMaster load balancer locally as well as tested against the [AWS Marketplace AMI](#).

Initial Look at LoadMaster



The API lives at the endpoint “/access”. A request to this endpoint is processed by a Bash script. The Bash script calls the “access” binary at “/bin/access” to set up permissions and then process the API request.



This image shows the Bash script called when requesting “/access”. \$0 is the path which is requested and “/bin/access” is executed with arguments.

Reverse Engineering the Access Binary

The access binary is responsible for handling the REST API requests. We decompiled the binary using Ghidra to understand what happens when the binary is executed. Starting in the main function we can see how a call to the endpoint “/access” is handled.

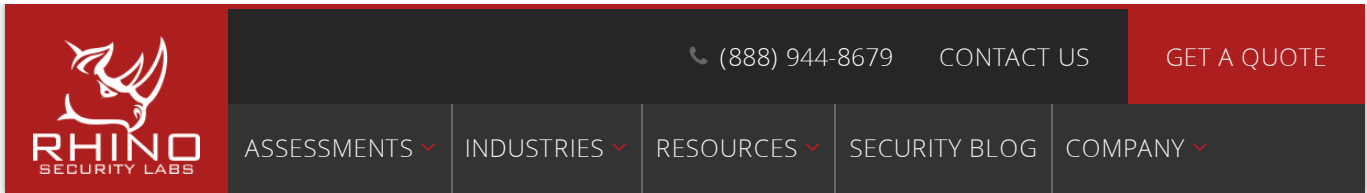


1. First the code tries to locate the API command [1] which is the second part of the path. For example, in a call to “/access/set” “set” would be the command.
2. The next block [2] performs some checks on the command and parameters and returns a 404 if they are not as expected.
3. Taking these points into consideration a path like: “/access/set?param=enableapi&value=1” should pass these checks and reach the “verify_perms()” function [3].

This would also allow the verify_perms() function to be reached even if the API is disabled in the application (disabled being the default setting).

Below you can see a normal request to “/access” with the API disabled. The request fails and returns a 404 as expected.





Now that we confirmed we can reach the “verify_perms()” function, let’s see what it does.

Within the “verify_perms()” function there is a call to “v1getuser()” which handles looking up the user making the request for the v1 API endpoint. Within “v1getuser()” these next two images show that the “REMOTE_USER” [1] and the “REMOTE_PASS” [2] environment variables are retrieved and then passed into the “validu()” [3] function.



The “REMOTE_USER” and “REMOTE_PASS” environment variables are set when the basic authentication header is decoded by the httpd server. The “validu()” function is used to check if the user is valid based on those variables.

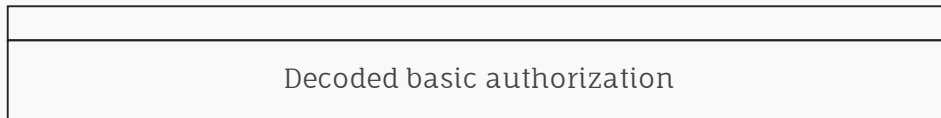
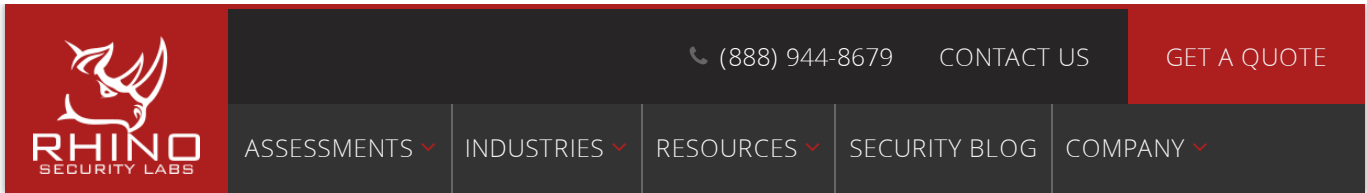
Next, looking at the “validu()” function we start to see something quite interesting. The “REMOTE_USER” and “REMOTE_PASS”, which can be *controlled by any unauthenticated user* by sending a basic authentication request, are used to construct a command with __sprintf_chk() [1] and then passed into a “system()” call [2] without any validation or sanitization.



At this point:

1. We can bypass the disabled API restrictions and reach it even if it is disabled.
2. We can supply unauthenticated user input, including special characters, as a basic authentication header.
3. That input is eventually evaluated by a “system()” call which executes commands.

Let’s see if this can result in command injection.



It worked as expected, injecting a command into the base64 encoded authorization string will then be executed by the server.

Impact

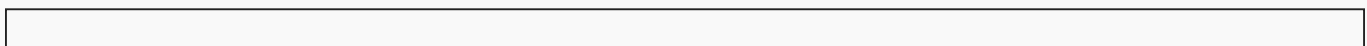
Exploiting this vulnerability enables command execution on the LoadMaster if you have access to the administrator web user interface. Once command execution is obtained, it is possible to escalate privileges to root from the default admin “bal” user by abusing sudo entries, granting full control of the device.

Proof of Concept

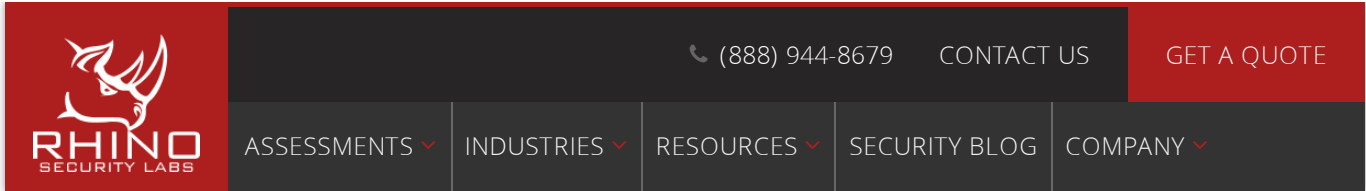
A proof of concept exploit is available in our [CVE GitHub repository](#). The GIF below shows the POC being used to read the boot message and patch version from the system.



We have also developed a [Metasploit module for this vulnerability](#).



Conclusion



Directly passing user input to system functions is strongly discouraged due to the significant security risks involved. If the operation is necessary, ensure that the user input is strictly validated and sanitized of any control characters prior to being passed to the function. It is imperative to conduct thorough reviews of both the code and applications in use as this exposes a critical attack surface.

Rhino wants to thank Progress software for working with us in quickly implementing a patch for this vulnerability.

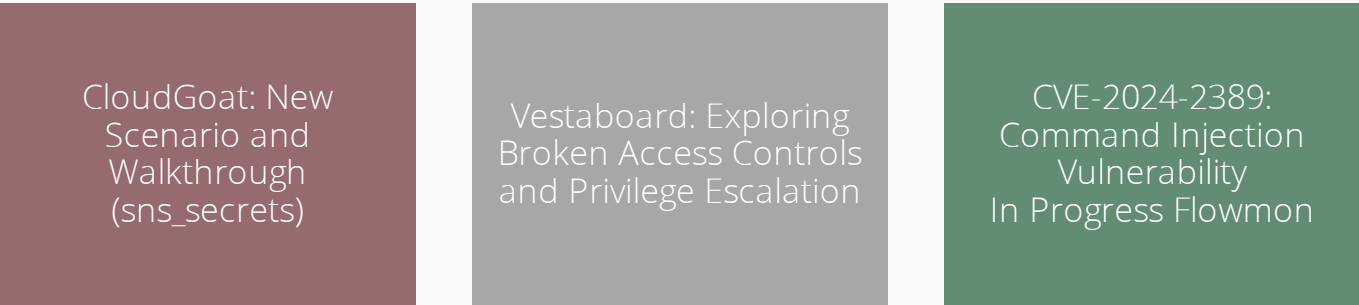
As always, feel free to follow us on Twitter or LinkedIn and join our Discord server for more releases and blog posts.

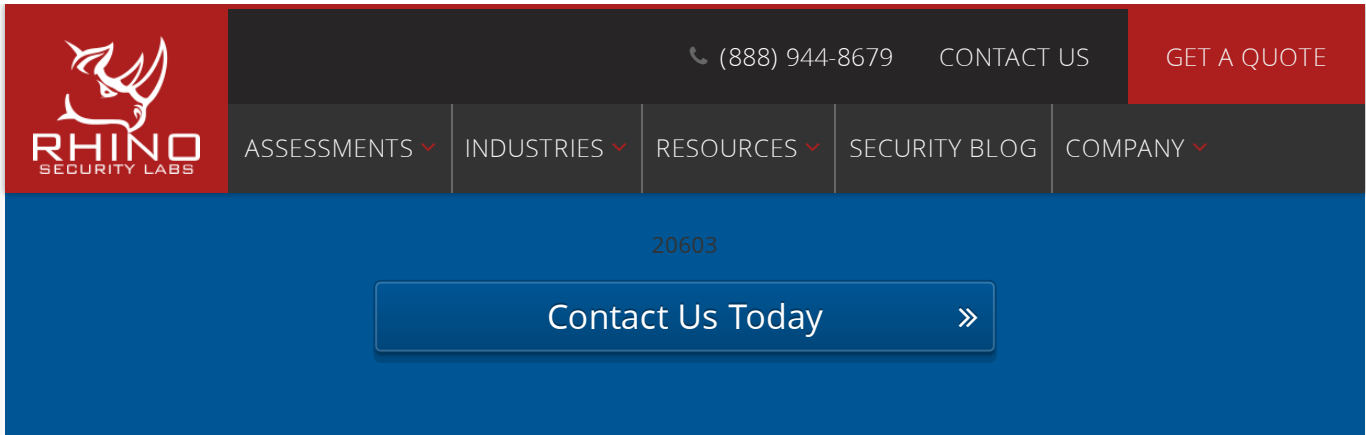
Twitter: <https://twitter.com/rhinosecurity>
LinkedIn: <https://www.linkedin.com/company/rhino-security-labs/>
Discord: <https://discord.gg/TUuH26G5>
Researcher/Author's Twitter: <https://x.com/daveysec>

Disclosure Timeline

1/31/2024	Issues reported to Progress Software
1/31/2024	Progress acknowledged the vulnerability and began working on a fix
2/8/2024	Progress assigns CVE-2024-1212 and publishes advisory and patch for customers
3/19/2024	Rhino published details on vulnerability

Related Resources





ASSESSMENT SERVICES

Network Penetration Test
Webapp Penetration Test
AWS Cloud Penetration Testing
GCP Cloud Penetration Testing
Azure Penetration Testing
Mobile App Assessment
Secure Code Review
Social Engineering / Phishing Testing
Vishing (Voice Call) Testing
Red Team Engagements

INDUSTRIES


Healthcare
Finance
Technology
Retail

RESOURCES

Technical Blog
Strategic Blog
Example Pentest Report
Technical Research
Vulnerability Disclosures
Disclosure Policy
Penetration Testing FAQ
Support: AWS Pentest Form

COMPANY

Leadership
Blog
Careers



[\(888\) 944-8679](#)

[CONTACT US](#)

[GET A QUOTE](#)

[ASSESSMENTS](#)

[INDUSTRIES](#)

[RESOURCES](#)

[SECURITY BLOG](#)

[COMPANY](#)

ABOUT US

Rhino Security Labs is a top penetration testing and security assessment firm, with a focus on cloud pentesting (AWS, GCP, Azure), network pentesting, web application pentesting, and phishing. With manual, deep-dive engagements, we identify security vulnerabilities which put clients at risk.

Endorsed by industry leaders, Rhino Security Labs is a trusted security advisor to the Fortune 500.

info@rhinosecuritylabs.com	(888) 944-8679	Rhino Security Labs, Inc	
--	--	--------------------------	--