APT

Glutton: A New Zero-Detection PHP Backdoor from Winnti Targets Cybercrimals

Alex.Turing, **Acey9**

2024年12月12日 • 13 min read

- **Introduction**
- Victims
- Analysis of Glutton
  - Modular Framework Design
  - Indicators of Glutton Infection
- Part1: task_loader
- Part2: init_task
  - 0x01: elf_install Task
  - 0x02: bt_modify Task
    - i. find_all
    - i. do_modify
  - 0x03: php_modify Task
- Part3: client_loader
  - Why Add a Backdoored Client?
- Part4: client_task
  - 0x01: PHP Backdoor
  - 0x02: fetch_task
- Easter Eggs in Glutton's Campaign
  - jklwang.com
  - HackBrowserData
- Conclusion
- IOC
  - MD5
  - C2
  - Downloader

Introduction

On April 29, 2024, `XLab's Cyber Threat Insight and Analysis System(CTIA)` detected anomalous activity: `IP 172.247.127.210` was distributing an **ELF-based Winnti backdoor**. Further investigation revealed the same IP had, on December 20, 2023, distributed a zero-detection malicious PHP file, init_task.txt, providing a key lead for the analysis.

Using `init_task` as a lead, we identified a series of associated malicious PHP payloads, including `task_loader`, `init_task_win32`, `client_loader`, `client_task`, `fetch_task`, and `l0ader_shell`. These payloads are highly modular, capable of functioning independently or being executed sequentially via `task_loader` to form a comprehensive attack framework. All code execution occurs within PHP or PHP-FPM (FastCGI) processes, ensuring **no file payloads are left behind, thus achieving a stealthy footprint**. This investigation uncovered a previously **undocumented advanced PHP backdoor**, which we named **Glutton** due to its ability to infect large numbers of PHP files and implant `l0ader_shell`. The core functionalities of Glutton include:

1. **Data Exfiltration**
    - System information, such as OS versions and PHP versions.
    - Sensitive Baota panel data, including credentials and management interface details.
2. **Backdoor Installation**
    - An ELF-based Winnti backdoor.
    - PHP-based backdoors.
3. **Code Injection**
    - Malicious code injection targeting popular PHP frameworks like Baota (BT), ThinkPHP, Yii, and Laravel.

The ELF sample `ac290ca4b5d9bab434594b08e0883fc5` that triggered the alert was delivered by Glutton's `init_task` component. This sample shares near-complete similarity with the PWNLNX tool discussed in BlackBerry's report "Decade of the RATs" and samples mentioned in IntezerLabs' September 23, 2020 tweet. Most security vendors currently classify this sample as a Winnti backdoor.

As a hallmark tool of the APT group Winnti, the Linux variant has not been observed in use by other hacking groups since its initial disclosure in 2019. The campaign's C2 server `156.251.163[.]120` remained active during the attack, properly responding to network requests and establishing interactions with the backdoor. This, coupled with the specificity of the sample and the C2's functionality, effectively rules out the possibility of interference from unrelated cybercriminal groups using dormant samples.

Key observations include:

- **Sample specificity**: The Winnti backdoor is a signature tool of the Winnti group, with no evidence of circulation among other cybercriminal entities.
- **C2 effectiveness**: The C2 server was fully operational, confirming the attack's authenticity.

Based on the veracity of the Winnti backdoor and Glutton's delivery mechanisms, it is theoretically plausible to attribute Glutton to the APT group Winnti. However, from a technical perspective, Glutton demonstrates several shortcomings in stealth and execution, which seem uncharacteristically subpar:

1. **Lack of encrypted C2 communications**: The protocol is overly simplistic and easy to reverse-engineer.
2. **Downloader communication over HTTP**: The lack of HTTPS makes traffic interception or monitoring trivial.
3. **Unobfuscated PHP samples**: The samples are in plaintext source code, making their functionality directly readable.

4. **Weak infrastructure deception**: The domain used (`thinkphp1[.]com`) is poorly disguised.

In summary, while Glutton's delivery mechanisms strongly align with the Winnti group, its lack of stealth and simplistic implementation introduce uncertainty. Attribution must account for the complexity of the cybercrime landscape and the inherent delays in defense-side intelligence. To avoid misleading conclusions based on isolated evidence, we adopt a conservative approach, attributing Glutton to the Winnti group with **moderate confidence** as a potential new weapon in their arsenal.

Victims

Infections caused by **Glutton** were identified through requests to its C2 server, `cc.thinkphp1[.]com` . Our analysis shows that victims were primarily located in China and the United States, spanning industries such as IT services, business operations, and social security.

"No Honor Among Thieves"

Interestingly, our investigation revealed that Glutton's creators deliberately targeted systems within the cybercrime market. By poisoning operations, they aimed to turn the tools of cybercriminals against them—a classic "no honor among thieves" scenario.

In July 2024, we conducted a VirusTotal hunt using the signature `"b11st=0;"` , which led to the discovery of five infected files uploaded from different countries:

INDEX	MD5	DETECTION	FIRST SEEN	COUNTRY
1	3f8273575d4c75053110a3d237fda32c	2/65	2024-08-11	China
2	c1f6b7282408d4dfdc46e22bbdb3050f	0/59	2024-09-17	Germany
3	96fef42b234920f3eacfe718728b08a1	0/63	2024-10-14	Singapore

INDEX	MD5	DETECTION	FIRST SEEN	COUNTRY
4	ad150541a0a3e83b42da4752eb7e269b	1/62	2024-11-02	United States
5	ad0d88982c7b297bb91bb9b4759ce0ab	4/41	2024-11-27	United States

Files 1–3 were standalone PHP scripts, while files 4–5 were archives containing full-fledged business systems. Of these, file 4 stood out as a fraudulent click-farming platform, a common tool in online scams. The malicious code, `l0ader_shell`, was embedded in the `APP.php` file of the ThinkPHP framework.

The VirusTotal analysis revealed that the parent archive was `shuadan109.timibbs.cc_20241026_175636.tar.gz`. This led us to its download page, where it was being sold for **980 USDT**.

The archive was hosted on **Timibbs**, a forum infamous for selling cybercrime tools and resources, including scripts for gambling, gaming, fake cryptocurrency exchanges and click-farming operations—all sold at premium prices.

While we didn't verify whether the VirusTotal sample perfectly matches the code sold on Timibbs (`980USDT felt like a poor investment, LOL`), the relationship between Glutton's creators and the forum appears to follow one of several possibilities:

1. **The hacker is a customer**, purchasing tools from the forum and embedding malicious code.
2. **The hacker breached the forum**, injecting backdoors into shared resources.
3. **The hacker collaborates with the forum**, co-developing compromised systems.
4. **The hacker operates independently**, with their tools later added to the forum.

Regardless of the details, one thing is clear: Glutton's authors exploited the cybercrime ecosystem itself, using poisoned tools to turn cybercrime operators into unwitting pawns. Their strategy might be best summarized like this:

> *"Why should these small-time scammers in gambling and click-fraud get all the money? Let's rob them blind! Here's the plan: flood the market with backdoored systems, let them unknowingly 'work' for us, and then cash out big-time. Even if they figure it out, they won't dare report it. Absolutely brilliant!"*

Analysis of Glutton

We have captured multiple components of **Glutton**, including `task_loader`, `init_task`, `client_loader`, `client_task`, `fetch_task`, and `l0ader_shell` (note: names like `client_loader`, `client_task`, and `fetch_task` are assigned based on their observed functionality). Each file contains approximately 3000 lines of code, none of which are encrypted or obfuscated, making their functionality relatively easy to analyze. This report will focus on the core functional code; readers interested in more details can refer to the full source code for deeper insights.

Modular Framework Design

These PHP components can operate independently or interact through `task_loader` as an entry point, incrementally loading other modules to construct a **fileless attack framework**. The framework's core capabilities include:

1. **Infecting PHP files** on the target device.
2. **Deploying backdoors**, including the Winnti backdoor and a PHP backdoor.

This modular design not only enhances the adaptability of the attack but also makes it harder to detect and trace during defensive operations.

We speculate that the attackers use multiple methods to spread Glutton, including:

- Exploiting traditional **0DAY and NDAY vulnerabilities**.
- Leveraging **weak password brute-forcing** techniques.
- Distributing pre-compromised business systems with embedded `l0ader_shell` via **cybercrime source code forums**, enabling targeted attacks on the cybercrime ecosystem itself.

Indicators of Glutton Infection

Infected devices exhibit the following signs:

1. **File-Level Indicators**:
   PHP files are injected with `l0ader_shell`.

2. **Process-Level Indicators**:
   - A **Winnti backdoor process** (`php-fpm`) listens on UDP port 6006.
   - A **PHP backdoor process** (`[kworker/0:0HC]`) communicates over UDP.

Part1: task_loader

The **task_loader** module plays a pivotal role in Glutton's attack chain. Its primary function is to assess the execution environment and use different methods to

download and execute the next-stage payload based on the detected environment. Key functions include:

1. `run_task_by_system`
2. `run&get_php_code`
3. `run_task_by_fpm`
4. `run_task_direct`

Functional Overview

The table below summarizes the behavior of each function:

FUNCTION	PATH	EXECUTION ENVIRONMENT
`run_task_by_system`	`/v11/init_task.gz`	New PHP process
`run&get_php_code`	`/v11/init_task.gz`	FastCGI
`run_task_direct`	`/v11/modify_php_v11.gz`	Original PHP process

Details of Payloads

1. `init_task`
   - Downloaded by both `run_task_by_system` and `run&get_php_code`.
   - Serves as the primary payload for further infection.
2. `modify_php`
   - Downloaded by `run_task_direct`.
   - A subset of `init_task`, optimized for specific modifications to the environment.

Part2: init_task

The `init_task` module performs three critical tasks:

1. `elf_install`: Downloads and executes the Winnti backdoor.
2. `bt_modify`: Infects Baota (BT) panels to collect sensitive information and modify system files.
3. `php_modify`: Infects PHP files to embed code for subsequent payload delivery.

0x01: `elf_install` Task

The `elf_install` task downloads the Winnti backdoor, masquerading it as `/lib/php-fpm`. To achieve persistence, it appends the following command to `/etc/init.d/network`:

```
export OLD=$PATH; export PATH=/usr/lib/; php-fpm; export PATH=$OLD;
```

Observed Download URLs and MD5

URL	MD5
`172.247.127[.]210/v10/php-fpm`	`ac290ca4b5d9bab434594b08e0883fc5`
`v6.thinkphp1[.]com/v11/php-fpm`	`ac290ca4b5d9bab434594b08e0883fc5`
`v20.thinkphp1[.]com/static/v20/php-fpm`	`ac290ca4b5d9bab434594b08e0883fc5`

The `ac290ca4b5d9bab434594b08e0883fc5` sample closely resembles the one exposed by BlackBerry, with additional functionality for updating C2 configurations and samples. The C2 configurations are encrypted with **rolling XOR** (key: `CB2FA36AAA9541F0`) and decrypt to: `156.251.163[.]120`

The IP has since become inactive, but historical evidence confirms it previously responded to Winnti network requests, indicating its role as a legitimate Winnti C2.

0x02: `bt_modify` Task

The `bt_modify` task targets Baota (BT) panels, performing two primary functions: `find_all` and `do_midify`.

find_all

Collects sensitive information, compresses and uploads the data to the C2 server.

ADMIN_PATH	BT_APASS	BASIC_AUTH	BASIC_PASS	BASIC_USER
bt_clients	crontabs	databases	bt_dir	bt_domain
bt_ftps	bt_https	bt_mobile	mysql_root	bt_pass_md5
bt_passwd	phpmyadmin	bt_port	bt_sites	bt_sites_path
bt_ssh	bt_user_md5	bt_username		

The traffic generated during this process is URL-encoded and compressed. Using tools like CyberChef (URL decode + raw inflate) allows for data reconstruction.

do_modify

Modifies critical BT panel files such as `init.py`, `public.py`, and `userlogin.py`, chieves objectives like: credential theft, token harvesting, exposing sensitive assets.

Key Modifications

- **Credential theft**: Inserts code to extract login credentials and tokens.

- **Asset exposure**: Alters configuration to expose sensitive assets.

0x03: `php_modify` Task

The `php_modify` task targets popular PHP frameworks such as ThinkPHP, Yii, Laravel, and Dedecms, injecting malicious code for further payload execution.

Modification Logic

- Searches for predefined `$ref_line` locations in the PHP framework code, inserts the `v11_code` at these locations.
- If no `$ref_line` matches, appends `v11_code` to the end of the file.

v11_code Structure

The `v11_code` consists of three parts:

1. `v11_begin`: `b11st=0;`
2. `PHPCODE_MAIN`: Encodes a `l0ader` function.
3. `v11_end`: `b11end=0;`

The `l0ader` function has two primary roles:

1. **Reporting**
   - Sends host information and page access parameters via UDP to `v6.thinkphp1[.]com:9988`.
2. **Downloading the Next-Stage Payload**
   - Constructs an HTTP request to download and execute the `client_loader` payload.

Traffic Analysis

The traffic generated during this process includes:

1. **UDP Traffic**: Transmits host and access information.
2. **HTTP Requests**: Retrieves the next payload (`client_loader`).

Part3: client_loader

The `client_loader` module is essentially a refactored version of `init_task`, retaining all of its core functionalities while introducing notable changes in code organization and additional features.

The first significant change lies in the `php_modify` task, where the `l0ader` function's code is now obfuscated, unlike its straightforward implementation in `init_task`.

The obfuscation adds a layer of complexity, making reverse-engineering more challenging for defenders.

The core functionality of the `l0ader` function remains unchanged; however, the network infrastructure used for communication has been updated.

MODULE	REPORTER	DOWNLOADER
`init_task`	`udp://v6.thinkphp1[.]com:9988`	`v6.thinkphp1[.]com/php?`
`client_loader`	`udp://v20.thinkphp1[.]com:9988`	`v20.thinkphp1[.]com/init?`

The most notable enhancement in `client_loader` is the introduction of a new capability: downloading and executing a backdoored **client**.

Why Add a Backdoored Client?

One might wonder why the attackers introduced a backdoored client when the Winnti backdoor was already deployed. The reasoning becomes clear when considering the broader objectives and the advantages of a PHP-based backdoor:

1. **Cross-Platform Compatibility**

- Unlike the ELF-based Winnti backdoor, the PHP client can operate seamlessly across Linux, Windows, and macOS systems.

2. **Fileless Payload Delivery**
   - By leveraging PHP for backdoor functionalities, the attackers achieve higher stealth through fileless execution, reducing the likelihood of detection.
3. **AV Evasion**
   - Antivirus engines often lack robust signatures for PHP-based malicious samples, allowing the PHP client to bypass traditional defenses.

Part4: client_task

The `client_task` module is responsible for two primary tasks:

1. Launching a PHP backdoor.
2. Periodically executing the `fetch_task` function to retrieve and execute additional payloads.

0x01: PHP Backdoor

The **PHP backdoor** functionality is implemented using the `client_socket` class, which provides a framework for backdoor operations.

Core Features

1. **C2 Communication**
   - Hardcoded C2: `cc.thinkphp1.com:9501`.
   - Supports both **TCP** and **UDP**, defaulting to UDP for communication.

2. **Command Execution**
    - The `client_v1` class extends `client_socket`, using the `process_std_cmd_v1` class to process commands from the C2 server.

3. **Supported Commands**
   The backdoor supports **22 distinct commands**, as shown below:

ID	FUNCTION
1	`ping` (UDP only)
2	`pong` (UDP only)
10	`login`
31	`keepalive`
148	`set connection config`
149	`switch connection to TCP`
150	`switch connection to UDP`
151	`shell`
152	`upload/download file via TCP`
189	`get_temp_dir`
190	`scandir`
191	`get dir info`
192	`mkdir`
193	`write file`
194	`read file`
195	`create file`
196	`rm`
197	`copy file`
198	`rename file`
199	`chmod`
200	`chown`
201	`eval PHP code`

Communication Protocol

- **UDP Communication**:
  - Includes an additional "liveness check" process with a `ping` from the client and a `pong` response from the server.
  - Typical interaction sequence: `ping → pong → login → cmd → heartbeat`.

- **Packet Structure**:
  - The first byte (`magic`) indicates compression, the second byte specifies the command code.
    - `0xf0`: No compression.
    - `0xf1`: Compression enabled (used for data >32 bytes).
- **Login Command**:
  - Contains host metadata such as `host_user`, `host_os`, `host_name`, and `host_cwd`. For compressed data (`0xf1`), the payload is parsed using "Raw Inflate"

0x02: `fetch_task`

The `fetch_task` function is executed hourly. It retrieves and executes additional PHP payloads by making an HTTP request to the remote server.

Payload Retrieval Process

- URL: `http://v20.thinkphp1.com/v20/fetch`.
- The response contains compressed PHP code, which is decompressed and executed.

Observed Payloads

Currently, the `fetch_task` function retrieves the `client_loader` payload, identified by the MD5 hash **69ed3ec3262a0d9cc4fd60cebfef2a17**.

Easter Eggs in Glutton's Campaign

jklwang.com

The `do_tp5_request` function in Glutton is used to clean up infections in older versions of the `Request.php` file. By analyzing the `$ref_lines` in the code, it was discovered that the domain `jklwang.com` (0 detections on VirusTotal) is also part of Glutton's infrastructure.

This suggests that Glutton's operators maintain a wider network of assets than initially detected, enabling them to extend their campaign reach.

HackBrowserData

On **June 14**, the domain `v20.thinkphp1.com` was observed distributing a **macOS version** of the HackBrowserData tool.

About HackBrowserData

A legitimate tool designed to decrypt and export browser-stored data, including: Passwords,Browsing history,Cookies, etc.

We hypothesize that HackBrowserData was deployed as part of a "black eats black" strategy. When cybercriminals attempt to locally debug or modify backdoored business systems, Glutton's operators deploy HackBrowserData to steal **high-value sensitive information** from the cybercriminals themselves. This creates a recursive attack chain, leveraging the attackers' own activities against them.

Conclusion

Based on the initial discovery of `init_task`, we estimate that **Glutton** has been active undetected in the cybersecurity landscape for over a year. In addition to targeting traditional “whitehat” victims through cybercrime, Glutton demonstrates a strategic focus on exploiting cybercrime resources operators. Its authors exhibit clear ambitions to **"win three times"**, reflected in the following:

1. **Stealing high-value sensitive information** from cybercrime operators.
2. **Profiting from the cybercrime industry itself**, leveraging infected systems for significant economic gain.
3. **Harvesting sensitive data** on crbercrime participants to enable future phishing or social engineering campaigns.

To mitigate the threat posed by Glutton, we recommend that system administrators take the following steps to identify and neutralize potential infections:

1. **Inspect all PHP files** for signs of `l0ader_shell`.
2. **Remove malicious processes**, including the Winnti backdoor process and the PHP backdoor process.
3. **Harden temporary directories** by creating a `.donot` file in `/tmp` to prevent exploitation.

This analysis represents the extent of our current understanding of the Glutton backdoor. Due to limited visibility, its **initial access vector** remains unclear. We invite contributions from partners and readers with relevant intelligence to help enrich the **technical and tactical matrix** of Glutton and improve attribution efforts.

If you are interested in our research, feel free to connect with us via Platform X to share insights or discuss collaborative opportunities. Together, we can work towards strengthening global cybersecurity.

IOC

MD5

```
17dfbdae01ce4f0615e9a6f4a12036c4  -  task_load
8fe73efbf5fd0207f9f4357adf081e35  -  init_task
8e734319f78c1fb5308b1e270c865df4  -  init_task
31c1c0ea4f9b85a7cddc992613f42a43  -  init_task_win32
722a9acd6d101faf3e7168bec35b08f8  -  client_loader
69ed3ec3262a0d9cc4fd60cebfef2a17  -  client_loader
f8ca32cb0336aaa1b30b8637acd8328d  -  client_task
00c5488873e4b3e72d1ccc3da1d1f7e4  -  v11_l0ader_shell
4914b8e63f431fc65664c2a7beb7ecd5  -  v20_l0ader_shell
6b5a58d7b82a57cddcd4e43630bb6542  -  modify_php
ba95fce092d48ba8c3ee8456ee4570e4  -  hack-browser-data-darwin-arm64
ac290ca4b5d9bab434594b08e0883fc5  -  winnti backdoor
```

C2

```
cc.thinkphp1[.]com
156.251.163[.]120
```

Downloader

```
IP
172.247.127.210


URL
v6.thinkphp1[.]com/php?
v20.thinkphp1[.]com/v20/init?
v20.thinkphp1[.]com/v20/fetch?
Reporter
```

```
udp://jklwang.com:9999
udp://{v6|v20}.thinkphp1[.]com:9988

http://{v6|v20}.thinkphp1[.]com/bt
http://{v6|v20}.thinkphp1[.]com/msg
http://{v6|v20}.thinkphp1[.]com/save
http://v6.thinkphp1[.]com/client/bt
```