Privilege Escalation

Performing and Fighting Privilege Escalation Attacks

Set-UID/Set-GID Hardening

Set-UID and Set-GID programs are dangerous because they give an ordinary user the privilege of another user, usually root.

If one of these programs has a vulnerability, the user can often force that program to extend their scope of use of that privilege!

Example: pmconfig

The pmconfig program had a vulnerability on it that an attacker once used on our system to attempt a privilege escalation.

While he was compiling, I realized that pmconfig didn't need to be Set-UID root or executable by non-admin users. I removed Set-UID before the exploit finished compiling.

Core Problem in Set-UID programs?

One of the major problems with these programs is that they're world-executable. This means that the web server user can run the, say, "traceroute" or "mount" programs when it really shouldn't be able to.

One major fix to this is to create a humans group.

Humans Group Maintenance

If the humans group seems to be annoying to maintain, create a quick script that can make sure that all users with UID>=x are in the group. We can set this to run every 1 hour / 1 day via cron.

x=500 on some distros, 1000 on others.

Challenge: can you write one in Perl or Python?

Sudo

Alternatively, use sudo to allow specific users to run commands with root privilege with complete logging.

They're asked their password on the first command and then periodically afterward.

You could use the NOPASSWD flag to require no password.

Sudoers

Create groups of users:

```
User_Alias      ADMINS = alice, bob
Host_Alias      INT = 192.168.1.0/24
Cmnd_Alias      CMDS = /bin/d,/bin/c

# Users         Hosts=Commands
ADMINS          INT = /usr/bin/wall
ADMINS          INT = NOPASSWD: CMDS
```

Set-UID Hardening Ref Card

Here's a reference you can use on your own machines:

```
find / -perm -04000 -uid 0 -ls > Set-UID-root-programs
find / -perm -02000 -gid 0 -ls > Set-GID-root-programs
```

You might use -xdev to lock to one mounted filesystem.

If you don't recognize something, learn about it.

Exercise

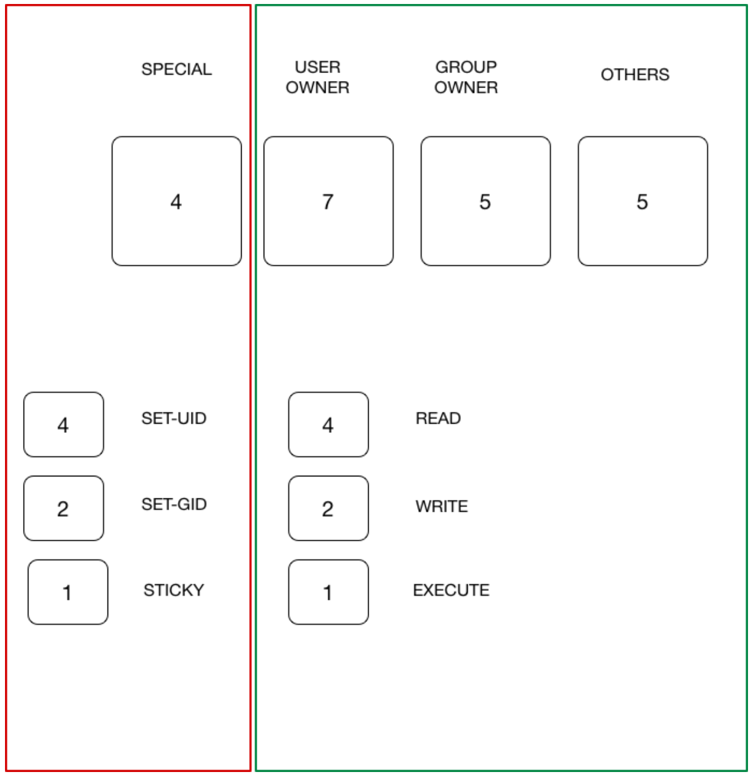
Please:

Open the Firefox browser on the class machine to:
http://localhost:10000/exercises/mrrobot-privesc

Permission Bits

755
User can read, write, execute
Group can read and execute
Others can read and execute

4750
Set-UID bit on
User can read, write, execute
Group can read and execute
Others can do nothing



Privilege Escalation through File Permissions

As an attacker who has landed a non-root shell on a system, weak file permissions are an excellent vector for escalating privilege.

If the attacker's user can write to any of these files, they've got root:

- Root's crontab file or any system crontab file (/etc/crontab, /etc/cron.hourly/* , …)
- Root or the system's shell configuration files (.bashrc, .profile, /etc/bash.bashrc,...)
- Any systemd unit file
- Any always/often-running system daemon's binary

If I can write data to a file that a root-running program will read, I may find a way.

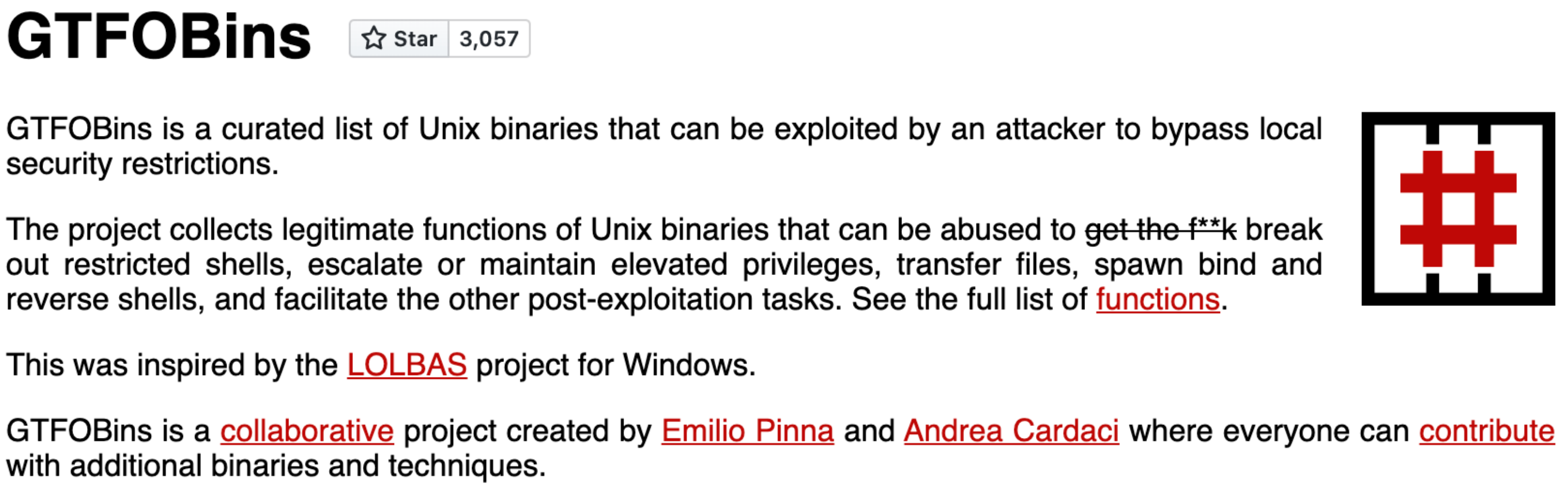
Abusing Sudo or Breaking out of Restricted Shells

If the user your shell runs as can run a Set-UID/sudo-enhanced command which runs other commands, you have a path to root. The same technique works for “breaking out” of a restricted shell like rbash. Here are a few examples:

Command	**Method of Getting Execution**
more, less, man, gdb, ftp	Start program, then type: !/bin/sh
vi or vim	Start program, then type: :!/bin/sh
scp	scp -S /tmp/yourscript file dest:
awk	awk ‘BEGIN { system(“/bin/sh”) }’
find	find / -name “somefile” -exec /bin/sh \;
tcpdump	tcpdump –i eth0 -w /tmp/f -W 1 -G 1 -z /path/to/yourscript

python	import os
	os.system('/bin/sh’)

GTFOBins



GTFOBins

GTFOBins is a curated list of Unix binaries that can be exploited by an attacker to bypass local security restrictions.

The project collects legitimate functions of Unix binaries that can be abused to ~~get the f**k~~ break out restricted shells, escalate or maintain elevated privileges, transfer files, spawn bind and reverse shells, and facilitate the other post-exploitation tasks. See the full list of functions.

This was inspired by the LOLBAS project for Windows.

GTFOBins is a collaborative project created by Emilio Pinna and Andrea Cardaci where everyone can contribute with additional binaries and techniques.

https://gtfobins.github.io/#+shell

Bonus Exercise

We'll have restricted shell breakout during exercises, but you can also get a bonus exercise for this.

After this class is over, you can download the vulnerable virtual machine named "Unknown Device" from the class website. It will give you additional practice in breaking out of restricted shells.

Permissions Hardening

We can do a great deal to keep weak file permissions from allowing easy privilege escalation.

At the very least, we should look for world-writable files and directories.

```
find / -perm -002 -type f -ls > ww-files
find / -perm -002 -type d -ls > ww-dirs
```

We can also look for files owned by system users:

nobody, web, dns, mail, ftp

Default File Permissions

The permissions on files that we create are 0777/0666, unless we have a umask that sets more restrictive permissions.

Add this line to the global shell configuration files:

```
umask 077
```

	File Locations
bash	`/etc/profile`
csh/tcsh	`/etc/csh.login`