DTU Compute
Department of Applied Mathematics and Computer Science

More than 20 years of Shellshock

Alberto Rico Simal

Andrew Habib

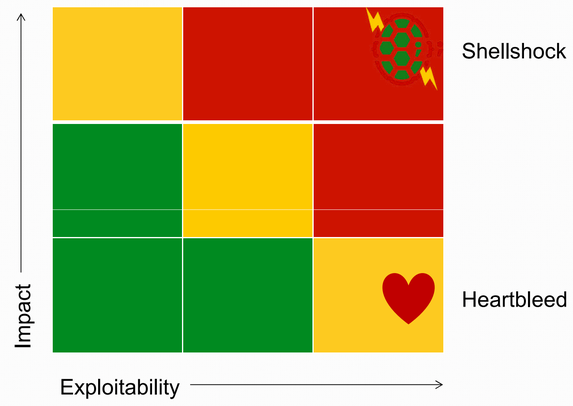
Dheeraj Kumar Bansal

{s135555,s135552,s135551}@student.dtu.dk

(1) What is Shellshock?

Shellshock is the series of security vulnerabilities discovered in the UNIX Bash shell, in late September 2014, including a whole array of exploits that allow attackers to gain remote shell and execute arbitrary code on affected devices. The importance of this security bug is evident due to the wide use of Bash in UNIX systems, which in turn represent the vast majority of server systems worldwide (estimated around 67%).

This, combined with the time this issue is considered to have been in the code base, has granted this vulnerability a rating of 10 out of 10, meaning it's considered to be of maximum impact and maximum likelihood.



To understand the significance of this rating, another recent bug that acquired similar notoriety was Heartbleed, affecting the negotiation of SSL connections, and potentially leaking data from the server's memory. For the Heartbleed, risk rating was just 5 out of 10. And turned out that Shellshock is incomparable to the Shellshock.

(2) The one-liner example

Shell shock is an Arbitrary Code Execution (ACE) vulnerability. A very easy and straightforward example on how this works is:

```
env foo='(){echo not patched;}' bash -c foo
```

First, declare an arbitrary environment variable with it's content being a function with empty name. Then call bash, and indicate it to run the string that it receives (in this case the reference to the environment variable, which it runs as a function.

The output of this script is "`command not found`" on a repaired system, but it is "`not patched`" on vulnerable Bash shells. This happens because unexpectedly, Bash automatically parses user-defined functions that are by default imported through environment variables. The real problem is with the context at which this function is invoked. To put it differently, Bash should not automatically import and parse arbitrarily user-defined environment variables and execute whatever commands written within those variables declaration.

(3) Impact

It is a 10/10 exploitable bug with 10/10 severe impact…

- **Up to today, 6 CVE (Common Vulnerabilities and Exposures)**
- **How many different classes of devices run Bash shell**

  computers, servers, routers, switches, firewalls, …
- **Bash is used in almost all Unix-like operating systems**
- **Examples of exploitable systems:**

  web applications that use CGI, DHCP servers (consequently the DHCP clients), …
- **Using the exploit for Reconnaissance attacks (83% of all attacks)**
- **Taking full control over servers (8% of all attacks)**
- **Utilizing the exploit for a very simple DDOS attack**
- **Create Botnets**

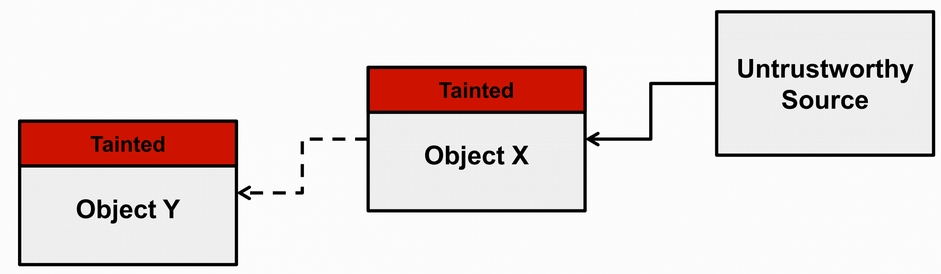
  "Wopbot' Botnet was quickly developed and is actively scanning and infecting exploitable systems
- **Around 1.5 million attack per day trying to use the bug**
- **Many other vulnerabilities could have been introduced into unpatched systems using the Shellshock bug**

(4) How could have we avoided it

Static Analysis

Information Flow Techniques: Taint Analysis

When an object receive data from untrustworthy source (like user input for example), then this object should be tainted (flagged or labeled so as to allow tracking of this object through the system to measure its influence. And this tainting is propagated through out the entire system in the sense that when a tainted variable is used to directly or indirectly derive the value of another variable, then the derived variable is also tainted.



Coding Practices

Use of Namespace

This was the solution ultimately introduced by most of the patches released to mitigate the breach. It has always been a good practice to use namespaces when using a shared resource – like the environment variables. A separate namespace should be created for each program and also whenever the program is using a large list of values. One could just have used the prefix `BASH_` for the importing function definitions.

Explicit data and/or code import

It is not always a good practice to by default enable implicit data and code import. Since this Bash functionality of importing functions through specially-formatted user-defined environment variables is rarely used, it should have been made available by explicit request only. In fact, an effective patch for the Shellshock was developed by enforcing a specific explicit request for functions import.

Software Documentation

Not only functionalities, but how they work

If the functionality of importing functions through user-defined environment variables and how it works have been well-documented and in details, the Shellshock bug could have been discovered much earlier.

(5) References

http://w3techs.com/technologies/details/os-unix/all/all

http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6271

http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160

http://www.dwheeler.com/essays/shellshock.html

http://blog.cloudflare.com/inside-shellshock/

http://stephane.chazelas.free.fr

DTU Compute
Richard Petersens Plads
Building 324
DK-2800 Kgs. Lyngby

+45 45 25 30 31
compute@compute.dtu.dk
www.compute.dtu.dk/english