International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 4, No.3 (July-2015)
http://dx.doi.org/10.12785/ijcds/040303

Implementation of SlowDroid: Slow DoS Attack Performed by a Smartphone

Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello

National Research Council, CNR-IEIIT
Via De Marini, 6 – 16149 – Genoa, Italy

Received 23 Nov. 2014, Revised 19 Jan. 2015, Accepted 7 Mar. 2015, Published 1 July 2015

Abstract: In terms of capabilities, today's smartphones are comparable to desktop computers. Last generation cellphones are indeed able to execute almost all the operation a common computer is able to accomplish. In this paper we focus on the use of mobile devices for perpetrating cyber attacks. With the purpose of proving the ability of executing attacks from a mobile environment, we introduce a mobile threat, SlowDroid, running on Android devices. SlowDroid implements a Denial of Service attack. Since it makes use of tiny amounts of resources, it is particularly accustomed to a mobile environment. We exhaustively present SlowDroid implementation and choices in terms of design, user interface and system architecture.

Keywords: android, mobile attack, cybersecurity, slow dos attack, denial of service

1. INTRODUCTION

In the last years, an emerging market appeared in the computer industry, relatively to Internet ready devices such as smartphones and tablet. The big companies of the Internet, such as Google, Apple and Microsoft are effectively in conflict for the conquest of the mobile market. Many companies and start-up are widely investing resources on the mobile market, announcing mobile oriented software and hardware with even more powerful capabilities. In this context, the last generation of devices is equipped with high performance hardware, such as processors, memory drives, sensors, localization chips, and different connection modules. In virtue of this, smartphones are currently able to accomplish operations not even thinkable before. This evolution has made mobile devices effectively able to perform almost every activity associated to desktop computing.

If we explore menaces involving mobile devices, they are often executed to target cellphones instead of exploiting them for perpetrate cyberattacks. Indeed, attackers usually inject malware, trojans, or viruses on the device to gather some kind of sensitive information or create a damage to the user. Historically, the first attack against cellphones arrived in 2000. The attack is commonly known as the Timofonica worm, designed to send SMS text messages to randomly generated numbers [24]. In the arena of attacks to mobile devices, it is only with the advent of the smartphone era that a wide variety of threats effectively appear. Some examples are the CommWarrior worm for Symbian OS [25] or the FakePlayer malware for Android operating system [32]. It is also worthy of mention the smudge attack, a "physical" threat able to detect unlock patterns by analyzing the smudges on touch screen surfaces [8].

Mobile devices always represented a target for attackers. Nevertheless, they have rarely been used as an attack tool. This paper focuses on the adoption of mobile and smart devices for perpetrating cyberattacks. A test case, we introduce the SlowDroid attack [11] running over the Android operating system. SlowDroid implements a Denial of Service (DoS) attack. Such threats are executed to make a network service unavailable on the network. The attacks belongs to a specific emerging category of DoS attacks: the first generation of such threats works by either exploiting a particular service or flooding the victim with a large amount of data. Differently, novel Slow DoS Attacks (SDA) [10] make use of tiny amounts of network bandwidth and computational resources. Because of this, we believe SDA are particularly suitable to a mobile environment.

The rest of the paper is organized as follows. Section 2 reports the related work of current threats. Section 3 motivates the advantages of a mobile threats execution.

E-mail: enrico.cambiaso@ieiit.cnr.it, gianluca.papaleo@ieiit.cnr.it, maurizio.aiello@ieiit.cnr.it

Section 4 reports the implementation of SlowDroid, while Section 5 describes how the attack works. Finally, Section 6 reports the conclusions of the paper.

2. Related Work

In this section we cover the related work on the topic. First, we introduce current Slow DoS Attacks in general, thus focusing on attacks perpetrated from mobile devices.

A. Slow DoS Attacks

Slow DoS Attacks represent the second generation of Denial of Service (DoS) menaces, after flooding based ones. Indeed, unlike the first generation of DoS attacks, SDAs make use of tiny amounts of bandwidth to lead a DoS on the victim. Attack bandwidth is reduced by working at the application layer of the ISO/OSI model, thus directly targeting the listening daemon running on the victim host. In fact, in comparison to the network/transport layer, at this layer the resources needed to overwhelm victim's resources are reduced. As a consequence of this, in the last few years, SDA emerged and consolidated as a dangerous attacks on the Internet.

Considering this category of attacks, most of the threats aren't related to research works, but they have been directly presented on the Internet, obtaining popularity and wide adoption.

Historically the first threat has been represented by the Shrew, an attack designed to send an attack burst to the victim, giving it the illusion of a high congestion on the network link [9]. Later, Maciá-Fernández et al. [13] introduced the Low-Rate DoS attack against Application Servers (LoRDAS) attack, leading a DoS on the victim by executing short attack bursts, focusing them to specific instants.

Instead, among the attacks born on the Internet, the most known threat maybe is Slowloris, a SDA implemented by Robert "RSnake" Hansen [29]. Like most of the slow DoS threat, Slowloris only targets the HTTP protocol. The attack works by establishing a large amount of pending requests with the victim and maintaining them alive as long as possible [21].

In 2012, together with the slowhttptest tool for executing a set of Slow DoS Attacks [14], the development team also introduced the Slow Read menace [26] for slowing down the responses of a web server by sending legitimate requests and specifying a small client-side reception buffer.

The Apache Range Headers attack has been published on the Internet as a script by a user known as "KingCope" [23]. This attack exploits the HTTP byte-range parameter, commonly used to request a portion of a resource, to force the server to replicate in memory a specific resource. Conversely to the previously mentioned threat, this attack should no longer be considered a menace [27], since appropriate patching systems have been deployed from Apache developers.

In this work we enrich the available set of SDA tools, proposing, executing, and analyzing the SlowDroid attack. The attack we propose works similarly to the Slowloris menace, since it is based on the same concept of constructing and sending to the server uncompleted requests. Nevertheless, in comparison to the other attacks mentioned, the proposed tool requires a minimum amount of attack bandwidth, thus making it particularly suitable to the mobile environment. Moreover, unlike most of the previously available attacks, the proposed menace can affect different protocols: messages payload is indeed not compliant to a specific protocol. Nevertheless, since custom payload is allowed, SlowDroid can also send specific well-formed messages affecting particular protocols. The proposed menace should therefore be considered a flexible tool, since it is able to lead to a DoS a wide variety of services.

Mobile Attack Tools

As previously introduced, until recently, due to their limited software, hardware and network resources, mobile devices were only considered a target for attackers, instead of exploiting them to accomplish malicious operations. Instead, mobile devices are nowadays able to accomplish operations comparable to ordinary computers. As a consequence, and also thanks to the advent of smartphones and last generation mobile operating systems (Android, iOS, Windows Phone, Firefox OS, etc…), apps developing is facilitated and malicious attacking tools are slowly reaching the mobile world too. In this paper we will now only consider Android operating system, since it represents the most common mobile operating system available [20] and due to the nature of the platform. Indeed, Android is considered an attractive operating system for hacking activities, due to the simplified app installation process, the possibility of obtaining administrator privileges on the device, and the open source nature of the system.

Considering the attacks deployed on the Android platform, various malevolent activities are covered. WiFiKill [31] is a tool used on shared wireless networks with the purpose of disabling Internet connection on specific devices.

DroidSheep [2] is instead an application for session hijacking, retrieving session cookies from hosts connected to the same network. This action would permit a malicious user to virtually impersonate the victim. A similar attack intercepting web session profiles is Faceniff [3]. These attacks represent a mobile porting of tools such as the Firesheep extension [4] for Firefox.

The mobile threats introduced above require administration privileges on the device. It is important to highlight that root privileges are available only on a small percentage of the devices, since it requires specific

procedures and knowledge owned only by a small portion of hackers. Therefore, this rooted device requirement, which is not needed in SlowDroid, represents an important limit.

Considering instead attacks that don't require administration privileges, various apps [16,15,18,17] are based on the Low Orbit Ion Cannon (LOIC) tool [28], a malicious software also adopted by the Anonymous group of hacktivists. The implementations differ one from the other. Nevertheless, they all make use of high amount of bandwidth and computational resources, since a flood against the victim is accomplished. In virtue of this, we believe such threats are not accustomed to a mobile environment, where resources are often limited.

An effective Slow DoS Attack implemented on a mobile platform is represented by the (unofficial) Slowloris mobile app [19]. Although the attack is able to successfully lead a DoS on the victim, it needs more attacking bandwidth than SlowDroid.

Therefore, the SlowDroid attack introduced in this paper should be considered an innovative tool designed to be executed on mobile platforms.

3. MOBILE DEPLOYMENT ADVANTAGES

As we have introduced above, the SlowDroid tool has been implemented as a mobile application running on Android operating system [11]. Android applications can be written in Java programming language. Since Java is a multi-platform language, it gives us the ability to reuse the same components on different (mobile or not) environments.

When designing a new software, it is important to choose a good environment accordingly to developers and final users needs. The environment is related to the technologies used during the development process, such as the programming language or the system architecture. In our case, the environment also includes the choice between a desktop/static or a mobile/dynamic execution of the tool.

From the attacker point of view, a mobile attack deployment may be preferred for many reasons. We will now briefly describe these reasons.

A. *Mobility*

Since mobile devices were born for communicating on mobility, they are carried out by people for the entire day. Therefore, a mobile execution of an attack offers the possibility to launch offensive operations from a wide range of places. Additionally, since smart devices are often equipped with several connection modules (Wi-Fi, 3G, LTE, Bluetooth, etc…), the attack is conveyed through one of these channels.

For example, we could imagine a user at the restaurant, exploiting the public Wi-Fi network to launch an attack against a particular victim, without being noticed by the other customers.

B. *Attack Hiding*

When running an attack in mobility, we could assume the attack is not interrupted in case the device (thus the attacker) is moving from a cell (like 3G, 4G, Wi-Fi, etc…) to another one. Particularly, in case of an horizontal or vertical handover [30], it is more difficult to detect and mitigate the attack, since the perpetrator's source address is continuously varying.

For example, we could imagine an attacker executing a malicious operation while cycling. In this case, the perpetrator would easily pass unobserved while executing the malevolent activity.

C. *App Spreading*

We can assume that a mobile deployment of a menace would represent an easy to use product. Indeed, if we consider last generation mobile operating systems, applications installation is considered an easy process, due to the user interface usability. Therefore, it is possible to easily reach large amounts of users with a simple publishing of a malicious tool (or an additional application embedding a malicious behavior). Moreover, if we consider the Android operating system in particular, the offered freedom allows users to install apps through third-party markets or directly from an Android Package APK files. Therefore, since no particular knowledge is required to install a malicious tool, almost every user owning a smartphone is able to install and execute an attack.

4. SLOWDROID IMPLEMENTATION

This section is focused on describing in detail the implementation of SlowDroid. The attack opens a specified amount of connections with the targeted server, with the aim of seizing all the service queue on the victim host. Under these conditions, a Denial of Service will be reached on the server and the adversary would maintain the DoS state during the attack execution time.

The SlowDroid tool we have implemented runs on Android based mobile devices and it has been published on the Internet [11].

We will now describe in detail the implementation.

A. *User Inputs*

In order to execute an attack/test, it is not required to master the Denial of Service topic. Indeed, the application has been designed to require to the user less information possible. Nevertheless, advanced configuration is possible, by customizing requests payload sent during the attack.

1) *Basic Configuration*

In order to target/test an Internet service, following basic information are needed:

- *Server IP Address* (String) identifies the IP address or domain name of the targeted/tested server;
- *Port* (int) identifies the listening port of the server's daemon;
- *Connections* (int) identifies how many connection will be simultaneously active during the attack execution;
- *Wait Timeout* (int) identifies, in seconds, the timeout used to alternate activity periods to idle ones (ON-OFF behavior).

2) Advanced Configuration

In order to execute an advanced attack, it is possible to customize the requests payload sent to the server. Following information can be customized:

- *Request Generation* (enum) identifies how requests payload is generated. Three possible values are allowed:
  - *Default*: requests are composed by a sequence of spaces;
  - *Random*: characters composing a request are chosen randomly, accordingly to the following regular expression: `[0-9a-zA-Z\-_ .,;:?/=*]`
  - *Custom*: a customized request format is used (accordingly to the next parameter).
- *Custom Request Format* (String) is used in case a custom request generation is adopted (otherwise, this parameter is ignored). This parameter identifies the custom request sent as payload during the attack.

3) Additional Configuration

It is also possible to set up additional settings:

- *Test Max Duration* (int) identifies, in seconds, the maximum duration of the attack.

This parameter has been introduced as SlowDroid has not been designed to be used for malicious operations. The concept behind the Test Max Duration setting is that an attack test can't be executed indefinitely and it will be sooner or later interrupted: the maximum duration for each attack is 3600 seconds. The same concept has been used to automatically interrupt an attack test in case the device's screen is turned off, or in case the application loses the focus. Indeed, the tool has been designed it to require a constant attention of the user.

B. Graphical User Interface

The tool has been implemented to maintain compatibility among a wide variety of versions of the Android operating system: the main Activity class implements PreferenceActivity [6], an Activity class implemented and available through the Android development framework, with the purpose of providing an easy to develop and extend interface for managing user preferences, compliant to the design of the entire operating system. This class allows developers to define a structure of preferences, thus automatically generating the user interface. Therefore, this choice allows us to reuse already available and consolidated software components.

For the same reason we provide to the user a menu that can be opened through the physical menu button on the device or by clicking an automatically shown menu button inside the application. Through the menu, users can launch a SlowDroid attack test with current settings, or obtain information about developers group. These two cases are treated similarly and a Dialog object [1] is opened. A dialog is a small window shown over the current activity and it is usually adopted for modal events.

The attack dialog is shown in Figure 1. As shown in figure, through the shown dialog, users can interrupt the attack at any time.

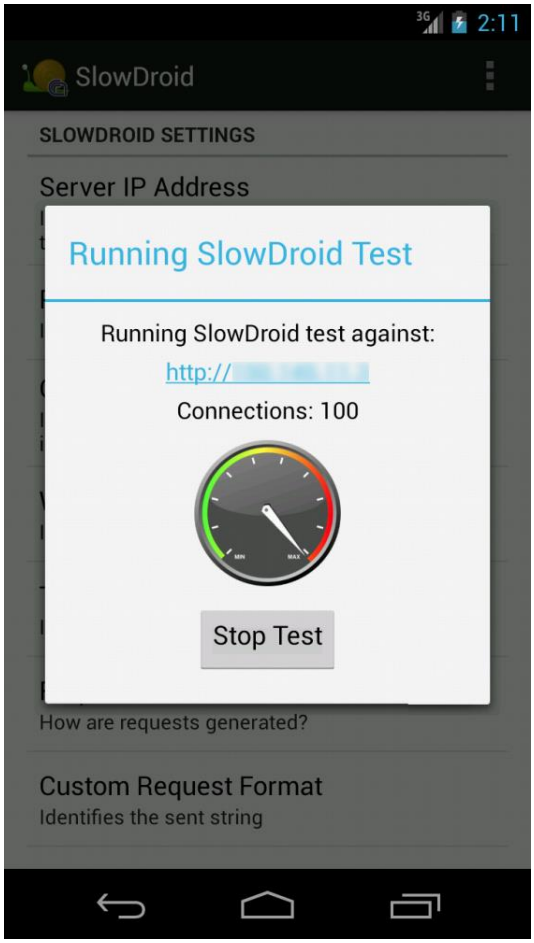

Figure 1. SlowDroid Attack Dialog

It is also shown the percentage of active connections over all the connections specified by the user. This value is shown as a percentage through a graphical odometer, represented with two different ImageView elements [5]. This percentage, along with the exact number of established connections, is updated every second. In order to enhance graphical results and provide a more realistic odometer, updates procedures are accomplished through a RotateAnimation object [7].

C. SlowDroid Library

With the purpose of making the attack reusable, mostly on a desktop environment, SlowDroid has been implemented in two different projects: the first one is represented by a Java attack library designed to instantiate a new attack and execute it. This library is not bounded to a mobile implementation; therefore it can be used on projects running on desktop environments as well. Instead, the second project is represented by the SlowDroid Android application. This application makes use of the attack library and implements the mobile application. Since Android applications can import Java libraries, the choice of splitting the project in two parts provides us the possibility of reuse of the implemented library on different projects, and at the same time maintaining compatibility with the Android application. As an example, the library may be (even stealthily) included in different (mobile or not) projects, with the aim of designing a new application from scratch, integrate the attack in a more exhaustive testing tool, or extending the attack itself through wrapping techniques.

5. ATTACK DESCRIPTION

The SlowDroid attack is based on the SlowReq threat [22], implementing an extended and enhanced version of the threat. SlowDroid exploits a vulnerability on most daemons implementations, often designed to limit the maximum number of simultaneous active connections to an extremely low value, with the aim of limit the number of connections simultaneously managed by the server. SlowDroid directly affects the application layer of the victim, trying to open with the listening daemon more connections than the ones it is able to manage. In virtue of this, in comparison to flooding based threats, less attack bandwidth is required. For this reason, the attack is particularly accustomed to the mobile environment.

We will now analyze the category of attacks which includes SlowDroid, describing how such attacks work.

A. Long Requests DoS Attacks

The category of Slow DoS Attacks which includes SlowDroid is known as Long Requests DoS [10]. Menaces of this type establish a large amount of connections with the server, sending uncompleted requests and saturating its resources while waiting for requests completion. Since requests are endless, this wait would result undefined.

1) Slowloris

The Slowloris attack [21] could be considered the most known Slow DoS Attack. Slowloris is a Long Request DoS that exploits the HTTP protocol. The attack works by establishing a specific amount of connections with the victim, as for others Long Request DoS threat. Additionally, through each connection, Slowloris sends the following specific data.

```
GET / HTTP /1.1\ r\n
Host: www.target.com\r\n
User Agent: Mozilla /4.0 [...]\ r\n Content -Length: 42\r\n
```

After receiving these data, the server's daemon would wait for the final `\r\n` characters, which identify the end of the request. Nevertheless, Long Request DoS attacks would never send such characters, thus forcing the server to an endless wait. Under these connections, a server side connection close would occur in case no additional characters are sent within a specific time period. With the purpose of maintaining the connections alive as long as possible, a Wait Timeout is used to periodically send a low amount of data and prevent closures. Particularly, Slowloris typically and repeatedly sends the following data, representing a single HTTP parameter.

```
X-a: b\r\n
```

2) SlowDroid

The behavior of SlowDroid is similar to the one of other Long Requests DoS such as Slowloris. Nevertheless, in this case different data payload is sent. Particularly, at any period, a single character is sent to the server. By default, a single space is always sent, but in general each character may be good. This behavior requires to the attacker a minimum amount of network bandwidth to induce a DoS on the server. Since the attack requires a minimum amount of bandwidth, it is particularly suitable to mobile environments, usually characterized by limited and expensive resources. Moreover, unlike most Slow DoS Attacks like Slowloris, in this case payload content is not compliant to a specific protocol. Therefore, SlowDroid can affect a wide range of TCP protocols (i.e. SMTP, FTP, etc…).

B. Effects of a SlowDroid Attack

The proposed SlowDroid tool is able to make an Internet service unreachable. From the server point of view, the effects of an attack execution are particularly interesting. Indeed, it often occurs that the server is unreachable just after a few seconds after the beginning of an attack, since all the available connections managed by the server are already (maliciously) seized.

Figure 2 shows the effects of a SlowDroid attack on a server. In particular, the number of connections established with an Apache2 web server without any protection module active is shown. Capture is relative to a duration of 600 seconds.

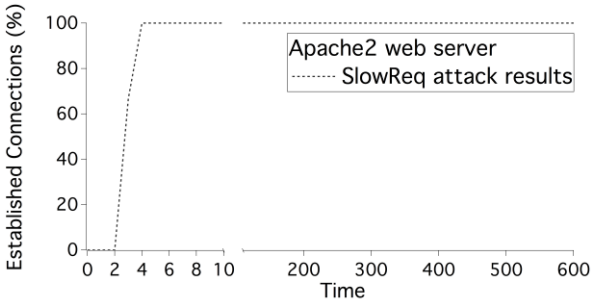

Figure 2. SlowDroid Effects on an Apache2 Web Server

It is possible to analyze that the DoS is reached on the server just after about 4 seconds. After this time, all the connections have been seized by the attacker and any additional (legitimate or not) connection is not able to communicate with the listening daemon until a connection slot is freed. In particular, although it's possible to connect to the server (at the transport layer), under such DoS an additional connection is not passed to the application layer, thus to the listening daemon, until a connection closure event occurs. Since slots are never freed during the execution of the attack, additional connections experience a DoS.

When launching a SlowDroid attack, some active connections may already be established with the server, from some other clients. Also in this case, the server would experience a DoS a few seconds after the begin of the attack. Nevertheless, each already seized connection would not be affected from the attack and it will be possible to communicate with the server through this channel. After a closure of the connection, the connection slot will be available and any additional connection will be allowed/accepted by the server. Actually, although there would be some sort of race condition with potential legitimate clients, the attacker would probably seize this connection as it becomes available, due to its intrusive behavior. For instance, SlowDroid has been implemented to detect a connection closure as soon as it happens and consequently re-establish the communication channel.

It is also important to mention that SlowDroid is particularly difficult to detect it while it is active. Indeed, log files on the server are often updated only when a complete request is received: in our case, requests are typically endless, and during the attack log files don't contain any trace of such a behavior. Therefore, a log analysis is not sufficient enough to produce an appropriate warning in reasonable times. Of course, as the attack proceeds and connections are closed due to some circumstances (forced reset, custom configurations on the server, timeout period occurrence etc.), the log files are updated.

C. Attack Functioning

As explained above, during a SlowDroid attack a certain amount of connections against the victim is established by the attacker. In case of a server without any already connected client, this number has not to be lower than the maximum number of simultaneous connections accepted by the server. Indeed, in this way all the available connections will be seized an a Denial of Service may potentially be reached.

Additionally, connections are kept alive by SlowDroid by periodically sending data (a single byte character is sent at any period), thus preventing closures.

The attack has been designed to execute three different program flows/threads for managing connections:

- the connect flow takes care of connections establishing, without sending any data to the server;
- the maintain flow maintains the connections with the server alive, by slowly sending data to the victim through the established channels, preventing server side connection closures;
- finally, the control flow identifies connections that have been closed by the server.

These three flows share a common variable that includes all active connections.

It follows a brief description of each flow with related code. We assume that a global `connectionsList[]` array variable is used, to include all the established connections that are active at a particular time.

1) Connect Flow

First flow's aim is to seize all the connections available at the application layer on the victim machine. Assuming that the maximum number of simultaneous connections the attacker wants to maintain alive with the server is m, this flow continuously check the currently established connections with the victim, opening the remaining ones, in order to reach the *m* value. It follows a representative execution procedure for this flow.

```
function ConnectFlow(int m) {
  while(True) {
    if(connectionsList.length == m)
      continue;
    connection = connect(host, port);
    if(connection != NULL)
      connectionsList.insert(connection);
    sleep(EPSILON); 7
  }
}
```

Note that the final `sleep()` call is accomplished in order to reduce computations on the application.

2) Maintain Flow

The maintaining flow takes care of maintaining the (already established) connections alive during the attack

execution. This thread makes use of the Wait Timeout parameter to manage the slowness of sending, accomplished through the waiting of the Wait Timeout expiration, thus sending a single character to the targeted server through the established channel. It follows a representative execution procedure for this flow.

```
function MaintainFlow() {
  while(True) {
    sleep(WAIT_TIMEOUT);
    foreach(connection in connectionsList)
      if(!connection.isActive())
        connectionsList.remove(connection);
      else
        connection.send('_');
  }
}
```

3) Control Flow

The control flow has the purpose of repeatedly check the status of the already established connections. This flow provides the attack the ability to re-establish closed connections as soon as they have been closed by the server. In particular, when a connection closure is identified by the control flow, it is removed from the `connectionsList` object. As a consequence, some instants later, this removal is detected by the connection flow, which would establish a new connection. In this way the attack is able to autonomously establish and maintain alive during the time *m* connections with the server. It follows a representative execution procedure for this flow.

```
function ControlFlow() {
  while(True) {
    foreach(connection in connectionsList)
      if(!connection.isActive())
        connectionsList.remove(connection);
      sleep(EPSILON);
  }
}
```

Assuming the *m* value is equal to the maximum number of connections accepted by the server, vulnerable to a SlowDroid attack, the attack is able to reach a DoS on the server. Moreover, thanks to the control flow, SlowDroid can successfully and quickly detect when the DoS is not reached anymore, thus trying to reach it again as soon as possible.

6. Conclusions and Future Work

In this paper we have introduced the SlowDroid Denial of Service testing tool. The tool implements an innovative attack that makes use of a tiny amount of bandwidth. Because of this, we have decided to develop SlowDroid on a mobile environment to demonstrate that even a single smartphone with limited capabilities is potentially able to lead a DoS on a corporate server. Since the tool has been implemented to be suited in a mobile operating system, we have also deeply analyzed and described the advantages of a mobile attack execution.

The published SlowDroid attack should be considered as a testing tool for system administrators willing to test the resilience of their servers to such attack. Although the tool is able to lead a DoS on a server, it is particularly easy to protect from a non-distributed menace such as SlowDroid [12]. Nevertheless, many servers on the Internet seems to be affected and unprotected from SlowDroid.

In order to avoid malicious and dangerous activities, we have deliberately reduced the functionalities of the published SlowDroid app: in particular, an extended and more dangerous version of the tool could have been published, in order to affect more server by specifying additional attack parameters and by implementing a distributed menace. Nevertheless, our purpose is not to deploy a cyberweapon, but to provide a testing tool and to prove that today's smartphones can be used as attack vectors. Because of this, the SlowDroid code has also been obfuscated, in order to hinder decompiling.

Further works on the topic may involve a porting of SlowDroid on different systems. In particular, since the attack has been implemented and is included in a separate Java library, a Java implementation aimed to execute the attack on different operating systems is facilitated. Moreover, it may be interesting to extend the tool to allow the execution of different attacks implementing a common interface.

It is also important to consider extensions needed from compatibility needs: since SlowDroid is bounded to the Android operating system, which is continuously evolving, future Android versions may require amendments to maintain compatibility.

References

[1] Dialog - Android Developers - Available at http://developer.android.com/reference/android/app/Dialog.html (Date Accessed on April 23, 2014)

[2] DroidSheep [ROOT] - Available at http://droidsheep.de/?page_id=263 (Date Accessed on April 23, 2014)

[3] FaceNiff - Facebook (and other services) Session Hijacker for Android - Available at http://faceniff.ponury.net (Date Accessed on April 23, 2014)

[4] Firesheep - Available at http://codebutler.github.io/firesheep/ (Date Accessed on April 23, 2014)

[5] ImageView - Android Developers - Available at http://developer.android.com/reference/android/widget/ImageView.html (Date Accessed on April 23, 2014)

[6] PreferenceActivity - Android Developers – Available at http://developer.android.com/reference/android/preference/PreferenceActivity.html (Date Accessed on April 23, 2014)

[7] RotateAnimation - Android Developers – Available at http://developer.android.com/reference/android/view/animation/RotateAnimation.html (Date Accessed on April 23, 2014)

[8] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, J. M. Smith: Smudge attacks on smartphone touch screens. Proceedings of the 4th USENIX conference on Offensive technologies pp. 1–7 (2010)

[9] A. Kuzmanovic, E. W. Knightly: Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications pp. 75–86 (2003)

[10] E. Cambiaso, G. Papaleo, G. Chiola, M. Aiello: Slow DoS attacks: definition and categorisation. International Journal of Trust Management in Computing and Communications - In press article (2013)

[11] E. Cambiaso, G. Papaleo, M. Aiello: SlowDroid Denial of Service tool for Android - Available at http://software.netsec.ieiit.cnr.it/projects/slowdroid/ (Date Accessed on April 23, 2013)

[12] E. Cambiaso, G. Papaleo, M. Aiello: SlowDroid Mitigation - Available at http://security.ge.cnr.it/?q=slowdroidmitigation (Date Accessed on April 23, 2013)

[13] G. Macia-Fernandez, J. E. Diaz-Verdejo, P. Garcia-Teodoro: Evaluation of a low-rate DoS attack against iterative servers. Computer Networks 51(4), 1013–1030 (2007)

[14] Google-Code: slowhttptest - Application Layer DoS attack simulator - Available at https://code.google.com/p/slowhttptest/ (Date Accessed on April 23, 2014)

[15] Google-Play: Loic - Available at https://play.google.com/store/apps/details?id=l.o.i.c (Date Accessed on April 23, 2014)

[16] Google-Play: LOIC - Low Orbit Ion Cannon - Available at https://play.google.com/store/apps/details?id=genius.mohammad.loic (Date Accessed on April 23, 2014)

[17] Google-Play: LOIC - Low Orbit Ion Cannon - Available at https://play.google.com/store/apps/details?id=genius.mustafa.loic (Date Accessed on April 23, 2014)

[18] Google-Play: PlexeDOS - LOIC - Available at https://play.google.com/store/apps/details?id=genius.plexe.loic (Date Accessed on April 23, 2014)

[19] Google-Play: Slowloris - Available at https://play.google.com/store/apps/details?id=com.kanuusan.slowloris (Date Accessed on April 23, 2014)

[20] H. McCracken: Whos Winning, iOS or Android? All the Numbers, All in One Place - Available at http://techland.time.com/2013/04/16/ios-vs-android/ (Date Accessed on April 23, 2014)

[21] ha.ckers: Slowloris HTTP DoS - Available at http://ha.ckers.org/slowloris/ (Date Accessed on April 23, 2014)

[22] M. Aiello, G. Papaleo, E. Cambiaso: SlowReq: A Weapon for Cyberwarfare Operations. Characteristics, Limits, Performance, Remediations. International Joint Conference SOCO'13-CISIS'13-ICEUTE'13 pp. 537–546 (2013)

[23] S. Alam: Apache released patch for ApacheKiller.pl Range Byte Flaw - Available at http://www.hackersgarage.com/apache-released-patch-for-apachekiller-pl-range-byte-flaw.html (Date Accessed on April 23, 2014)

[24] S. Coursen: The future of mobile malware. Network Security 2007(8), 7–11 (2007)

[25] S. Furnell: Handheld hazards: The rise of malware on mobile devices. Computer Fraud & Security 2005(5), 4–8 (2005)

[26] S. Shekyan: Are you ready for slow reading? - Available at https://community.qualys.com/blogs/securitylabs/2012/01/05/slow-read (Date 2012)

[27] SpiderLabs-Anterior: Mitigation of Apache Range Header DoS Attack - SpiderLabs Anterior - Available at http://blog.spiderlabs.com/2011/08/ mitigation-of-apache-range-header-dos-attack.html (Date Accessed on April 23, 2014)

[28] Wikipedia: Low Orbit Ion Cannon – Available at http://en.wikipedia.org/wiki/Low_Orbit_Ion_Cannon (Date Accessed on April 23, 2014)

[29] Wikipedia: Slowloris - Available at http://en.wikipedia.org/wiki/Slowloris (Date Accessed on April 23, 2014)

[30] Wikipedia: Vertical handover - Available at http://en.wikipedia.org/wiki/Vertical_handover (Date Accessed on April 23, 2014)

[31] xda-developers: Kill Wifi For Those Who Annoy You And Slow Your Network Down - WifiKill For Android - Available at http://www.xda-developers.com/android/kill-wifi-for-those-who-annoy-you-and-slow-your-network-down-wifikill-for-android/ (Date Accessed on April 23, 2014)

[32] Y. Zhou, X. Jiang: Dissecting android malware: Characterization and evolution. Security and Privacy (SP), 2012 IEEE Symposium on pp. 95–109 (2012)

Enrico Cambiaso graduated in Computer Science at the University of Genoa, Italy, in 2012, with a thesis entiled 'Analysis of slow DoS attacks'. He is a PhD student at the University of Genoa and he collaborates with the Research National Council of Italy, working to the slow DoS field. His scientific interests are related to computer and network security, intrusion detection systems, covert channels and cloud computing.

Gianluca Papaleo graduated in Computer Science at the University of Genoa, Italy in 2005. He is a Fellow Researcher of the National Research Council since 2006. His main scientific interests are in the field of computer and network security, intrusion detection systems, wireless communications and covert channels. His current teaching activity in University of Genoa are focused on wi-fi security and tunneling protocols.

Maurizio Aiello graduated in 1994, worked as a free-lance consultant in the field of system and network management both for universities and research centre and for private industries. From August 2001, he is responsible for the Region of Liguria of National Research Council network infrastructure. He is a Teacher of the course 'Network Security' at the University of Genoa and University College of Dublin; Coordinator of students, fellowships and EU projects in the field of computer security. He is involved in activities related to technology transfer (spin-off). His research activities are network security and protocols.