ARMY RESEARCH LABORATORY

# Application Analysis and Decision with Dynamic Analysis

**by Joshua S Edwards**

**ARL-CR-0754** **December 2014**

**prepared by**

**ICF International**
**9300 Lee Highway**
**Fairfax, VA 22031**

**under contract**

**W911QX-14-F-0020**

## NOTICES

### Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

**Army Research Laboratory**
Adelphi, MD 20783-1138

**ARL-CR-0754** **December 2014**

# Application Analysis and Decision with Dynamic Analysis

**Joshua S Edwards**
**ICF International**

**prepared by**

**ICF International**
**9300 Lee Highway**
**Fairfax, VA 22031**

**under contract**

**W911QX-14-F-0020**

| REPORT DOCUMENTATION PAGE | Form Approved OMB No. 0704-0188 |
|---|---|



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED (From - To) |
|---|---|---|
| December 2014 | Final | Oct 2013 – Oct 2014 |

| 4. TITLE AND SUBTITLE | 5a. CONTRACT NUMBER |
|---|---|
| Application Analysis and Decision with Dynamic Analysis | W911QX-14-F-0020 |
| | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |

| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
|---|---|
| Joshua S Edwards | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |

| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) | 8. PERFORMING ORGANIZATION REPORT NUMBER |
|---|---|
| ICF International<br>9300 Lee Highway<br>Fairfax, VA 22031 | ARL-CR-0754 |

| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
|---|---|
| US Army Research Laboratory<br>ATTN: RDRL-CIN-D<br>2800 Powder Mill Road<br>Adelphi, MD 20783-1138 | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

**12. DISTRIBUTION/AVAILABILITY STATEMENT**

Approved for public release; distribution unlimited.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

The US Army Research Laboratory currently has a mobile application analysis tool known as A2D, Application Analysis and Decision. It is a static analysis tool that takes in a mobile application and parses out a large amount of data relevant to human analysts and stores it in a database. It would then use some of these data to generate a risk score for the application. However, static analysis alone is not enough to fully analyze an application. To overcome the obstacles inherent in static analysis, A2D was extended to include dynamic analysis functionality. This will allow A2D to run an application in a controlled, virtual environment, and interact with it in ways similar to a human user in an attempt to elicit a response from the application. Once the series of interactions have concluded, A2D will gather useful data for analysis and store them in the primary A2D database for future analysis.

**15. SUBJECT TERMS**

Android, dynamic analysis

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| A. Report | b. ABSTRACT | c. THIS PAGE | | | Joshua S Edwards |
| Unclassified | Unclassified | Unclassified | UU | 22 | 19b. TELEPHONE NUMBER (*Include area code*)<br>301-394-1578 |

**Standard Form 298 (Rev. 8/98)**
**Prescribed by ANSI Std. Z39.18**

## Contents

## List of Figures



## List of Tables

## Acknowledgments

Special thanks go to Nathaniel Lageman and Mark Lindsey from Pennsylvania State University for initial development of the multiprocessing functionality between the host process and analysis pairs.

INTENTIONALLY LEFT BLANK.

# 1. Introduction

In 2012, the US Army Research Laboratory began development of a mobile application analysis tool known as A2D, Application Analysis and Decision. This was a static analysis tool that took in a mobile application and parsed out a large amount of data relevant to human analysts and stored it in a database. It would then use some of these data to generate a risk score for the application.

A2D is still in use, but it is missing a major feature; it is only a static analysis tool. The applications it processes are never actually run—leaving it vulnerable to obfuscation techniques that may hide the true functionality of the application.

To overcome this obstacle, work began on a dynamic analysis extension. This new functionality installs and launches the application on 1 of several virtual machines (VMs) that sit on top of a simulation of a standard network. The application will not be capable of reaching the wider Internet. As it runs, A2D will interact with the virtual phone and perform actions that the application may be waiting for, such as sending Short Message Service (SMS) messages and changing the phone's geographic location. These interactions will not be arbitrary; rather, they will be chosen based on metadata about the application collected during static analysis.

Once the series of interactions have concluded, A2D will gather useful data for analysis, including: logcat output, strace output, network traffic, and a screenshot. These data will be retrieved and stored in a primary A2D database for future analysis.

# 2. Structure

Nearly all of the dynamic analysis functionality takes place within a virtual environment. For this effort, a VMWare ESXi server was used. Several VMs run on this server and perform the dynamic analysis. These are shown in Fig. 1, with the following explanation:

1) Entry box: The box that is the entry point to the analysis process. It accepts the incoming applications, assigns the analysis to the virtual Android networks, and eventually writes the results to the database. For this instance, it is a Lubuntu 12.04 Operating System (OS) with about 8 GB Random Access Memory (RAM), 8 Central Processing Units (CPUs), and 450 GB of hard drive space.

2) Database: A MongoDB database that holds the results of static and dynamic analysis. This server is a VM merely as a manner of convenience and should be run on a separate

physical server in a realistic setting. More information on the data structure will be explained under Methods and Results.

3) Analyzers: Several of these VMs exist as clones of each other. They each have 2 network interface cards: one that connects to the primary VMWare network for communication, and another that connects only to the virtual mobile device assigned to that analyzer instance. These machines interface with the mobile devices for analysis and simulate regular network infrastructure, such as web and domain name servers. These VMs are all currently Lubuntu 13.10 OS with 1024 MB RAM, 1 CPU, and 20 GB of hard drive space.

4) Android VMs: Currently, these are all individual clones of an Android 4.0.3 device. They are all preconfigured to identify their respective analyzer as a domain name server. In the future, additional Android versions will be present and more steps will be taken to obfuscate their identity as VMs.

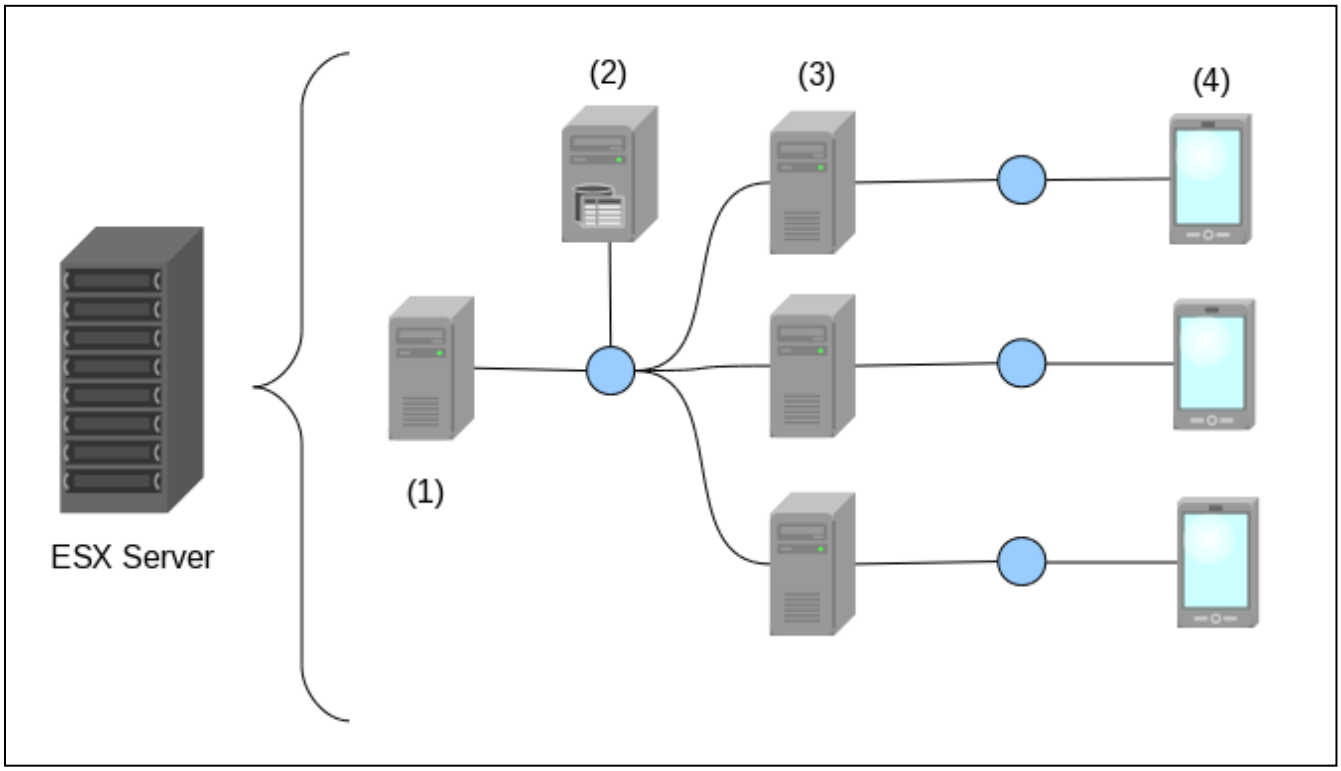


Fig. 1 Overview of VM network structure

The virtual network that connects the entry box, the database, and the analyzers is the default network that connects most machines, including the non-VM physical machines.

Each of the analysis networks between the analyzers and the Android VMs exists to capture traffic that moves between the phone and the analysis box that interacts with it. The devices on these networks each have specific Internet Protocol (IP) addresses, in keeping with their

existence as clones of each other. This IP setup ensures a degree of consistency with captured network traffic.

# 3. Methods

## 3.1 Entry Box

After an application has been statically analyzed by A2D and its results stored in the database, dynamic analysis can begin. The application is pushed to the entry box and the host process is launched. This host process collects the static analysis results from the database, chooses a free analysis pair, reverts them to pristine snapshots, and pushes the application and result metadata to the analyzer. It then launches the dynamic analysis scripts on the selected analyzer. The host process then enters a loop and waits patiently for the analysis pair to finish and return the results.

The host process is a Python script named “host_process.py”. During initialization, it spawns a separate producer thread that solely pushes applications into a shared queue. Then, the host process creates several consumer threads based on the number of analysis pairs available. These consumer threads constantly pull fresh applications from the shared queue and perform the actual dynamic analysis.

Before a consumer thread begins the dynamic analysis, it uses the pymongo Python module[1] to establish a connection with the database server and requests several points of data about the application from its static analysis results: permissions, phone numbers, domains, IP addresses, hashes, and the Android manifest file. The results are converted into a JavaScript Object Notation (JSON) file that will eventually be pushed to an analyzer.

The process attaches to the selected analyzer via Secure Shell (SSH) using the third-party paramiko Python module.[2] It then opens a Secure File Transfer Protocol (SFTP) connection and pushes the application file and the JSON file containing the metadata from the database.

When the 2 files are in place, the consumer thread starts the dynamic analysis script on the analyzer via the SSH connection.

As the dynamic analysis script runs on the analyzer, the thread enters a loop that constantly checks to see whether output files have been generated. This check is essentially a call over SSH of “ls /tmp/out.*” and a check for the existence of 3 of the expected output files: “out.pcap”, “out.strace”, and “out.logcat”.

The 3 output files represent:

- out.pcap: captured network traffic in the form of tcpdump packet capture
- out.strace: a log of system function calls using the strace tool

- out.logcat: the standard logging output for Android

When the 3 files are present, the consumer thread breaks out of the sleep cycle and continues processing. SFTP is used again to pull the 3 output files and attempt to pull the front end screenshot. Not all applications have a front end and thus will not have a screenshot present.

With the files available locally, the thread stops its analyzer and mobile VMs.

All of the files are prepared for insertion into the database. The screenshot, network traffic, strace, and logcat output files are directly inserted into the database, with the latter 3undergoing additional parsing.

First, the strace output is broken down. A parser takes each line and extracts the process identification (ID), function name, arguments, descriptor, timestamp, and any error codes present. A Python dictionary is assembled using these data and other implied metadata—such as the SHA256 hash of the application for ID, the identifier of the strace file in the database, and a run identifier to differentiate separate analysis attempts of the same application. Each of these dictionaries is pushed to the database for each line in the strace file for future analysis (Fig. 2).

```
{'_id': {'apk_id': 'b3ffca1691c948f397c85be4a751eb0f74a4b0506d1aa2af897c262c548cbb12',
         'datetime': '2014-04-02T18:50:28.861Z',
         'instance_hash': 'f3a4d50fbd3cbd53eb7903220c960589',
         'run_id': 0},
 'apk_id': 'b3ffca1691c948f397c85be4a751eb0f74a4b0506d1aa2af897c262c548cbb12',
 'args': ('/data/data/com.socialnmobile.dictapps.notepad.color.note/databases/colornote.db-wal",
F_OK)',
 'datetime': '2014-04-02T18:50:28.861Z',
 'descriptor': '-1',
 'error': 'ENOENT (No such file or directory)',
 'file_id': 'f6bc54cd326e7d7b6d323f422309d1e2d542384f8abcf8894b8580f848272533',
 'full_line': '[pid  1934] 1396479028.861210
access("/data/data/com.socialnmobile.dictapps.notepad.color.note/databases/colornote.db-wal",
F_OK) = -1 ENOENT (No such file or directory)',
 'function': 'access',
 'pid': '1934',
 'run_id': 0,
 'time': '1396479028.861210'}
```

Fig. 2 Example strace line after parsing

The logcat output is parsed next. It is handled very similarly to the strace output. Each line has the date, time, process ID, thread ID, logging level, grouping tag, and the actual log message extracted. Each line becomes a Python dictionary and is inserted into the database individually (Fig. 3).


```
{'_id': {'apk_id': 'b3ffca1691c948f397c85be4a751eb0f74a4b0506d1aa2af897c262c548cbb12',
         'datetime': '2014-04-02T22:46:21.976Z',
         'instance_hash': '2e8fcfb85a95807b86d0a712cefc79dc',
         'run_id': 0},
 'apk_id': 'b3ffca1691c948f397c85be4a751eb0f74a4b0506d1aa2af897c262c548cbb12',
 'datetime': '2014-04-02T22:46:21.976Z',
 'file_id': '195bbacf4b2b1f274d712d541357e4672071e622707def29838264101a93ae87',
 'full_line': '04-02 22:46:21.976  1269  1281 W PackageParser: Unknown element under <intent-
filter>: intent-filter at /system/app/AndAppStore-1_6_9.apk Binary XML file line #28',
 'level': 'W',
 'message': 'Unknown element under <intent-filter>: intent-filter at /system/app/AndAppStore-
1_6_9.apk Binary XML file line #28',
 'pid': '1269',
 'run_id': 0,
 'tag': 'PackageParser',
 'tid': '1281',
 'time': '22:46:21.976'}
```


Fig. 3 Example logcat line after parsing

Finally, the packet capture file is read. The parser uses the dpkt Python module to process the capture file one packet at a time.[3] Every packet has the following pulled, if applicable: the source and destination IP addresses, source and destination ports, timestamp, and data. These values are placed into a Python dictionary, similar to strace and logcat output, and stored in the database for each packet.

Packets for some protocols have additional pieces extracted. Domain name lookups have the requested domain included in the value dictionary, whereas web requests have the Hypertext Transfer Protocol (HTTP) headers and lookup method included (Fig. 4).


```
{'_id': {'apk_id': 'eec4ade3830652ef07baf5081727403818bea47e0f864dfdc93352f1b5659aa2',
        'datetime': '2014-04-02T18:50:54.457Z',
        'instance_hash': '2cac8ed1d0ff2095dcaaede5f6d560f6',
        'run_id': 0},
 'apk_id': 'eec4ade3830652ef07baf5081727403818bea47e0f864dfdc93352f1b5659aa2',
 'data': 'afMBAAABAAAAAAAAA3d3dwhmYWNlYm9vawNjb20AABwAAQ==',
 'datetime': '2014-04-02T18:50:54.457Z',
 'dip': '192.168.1.102',
 'dip_int': 3232235878L,
 'dns_lookup': 'www.facebook.com',
 'dport': 53,
 'file_id': '25b5bcba8863ab96994de5dca9b02b4bcdc46256b2489e784492ad7796291350',
 'instance_hash': '2cac8ed1d0ff2095dcaaede5f6d560f6',
 'protocol': 'UDP',
 'raw_packet':
'CAAnlcbqCAAnj/JMCABFAAA++iZAAEARvGrAqAFnwKgBZm0aADUAKq4eafMBAAABAAA
AAAAAA3d3dwhmYWNlYm9vawNjb20AABwAAQ==',
 'run_id': 0,
 'sip': '192.168.1.103',
 'sip_int': 3232235879L,
 'sport': 27930}
```


Fig. 4 Example packet, performing a domain lookup, after parsing

Once the last of these data is stored in the database, the host process is finished. The data have been collected, stored, and will await further analysis and assessment.

### 3.2 Analyzer

On the analyzer machines, several services run that simulate a larger network. Two simple Python-based web servers run on ports 80 and 443. These are direct command-line calls to the stock Python module SimpleHTTPServer. Each point to a directory contains only 1 Hyper Text Markup Language (HTML) document. They are meant to encourage applications to continue and not send a reset signal when requesting a web resource. A domain name system (DNS) server is also running, which redirects all lookups to the analyzer's IP address. This DNS server will ensure that all traffic is seen going toward the same box. It is implemented by calling the third-party simpledns Python module by JD Zamfirescu, derived from a script by Francisco Santos, and letting it run as a service. All of this is to avoid allowing a malicious application to break out of the sandbox into a proper network.

The primary dynamic analysis script lives on the analyzer. Once the host process pushes the application and metadata to the analyzer, the host launches this script to begin communications between the analyzer and its paired mobile device.

Before pushing the application to the phone, the script sets some fake values that it may provide the mobile device over the course of analysis. This information includes a randomly generated phone number for attempted phone calls and SMS messages and a random latitude–longitude location to simulate phone movement.

Next, the script reads the JSON-formatted metadata provided by the original host process. These data are used later to determine which actions are relevant and should be attempted over the course of running the application. For example, if an application requires permission to access a phone's location, the script will change the location several times to elicit a reaction from the running application.

The script now begins to interact with the mobile VM. It first establishes a connection using Google's debugging tool, "adb". The tool, adb, is used several times to interact with the mobile VM, by capturing the screenshot, sending SMS messages, executing arbitrary shell commands, and installing the application.

With the connection established, calls are made to start tcpdump for reading network traffic and logcat to watch log files.

Analysis is ready to begin in earnest. The application is installed on the phone and then launched, all via adb. After a delay to allow the application to warm up, a screenshot is taken of the initial interface, if applicable.

What follows is a series of artificial interactions that are called depending on requested permissions. Table 1 shows which requested permissions result in which interactions. An application only needs to request 1 of the permissions to cause that row's actions to occur. Unless otherwise indicated with a lowercase prefix, assume all listed permissions begin with "android.permission".

Table 1 Permissions and associated interactions

| Permissions | Analysis interactions |
|---|---|
| ACCESS_FINE_LOCATION<br>ACCESS_COARSE_LOCATION<br>ACCESS_GPS<br>ACCESS_LOCATION<br>ACCESS_LOCATION_EXTRA_COMMANDS | Creates a telnet connection to the phone and sends a command to change the geolocation of the phone using the arbitrary latitude–longitude previously generated |
| SEND_SMS<br>READ_SMS<br>WRITE_SMS | First, interacts with the phone to send a random SMS message from the phone to the previously generated false phone number. Then, the script makes a telnet connection to send another random text message to the phone. |
| READ_CONTACTS<br>WRITE_CONTACTS | Adds a contact to the phone's local contact list consisting of a randomly generated name, email address, and the previously generated false phone number |
| ACCESS_WIFI_STATE<br>CHANGE_WIFI_STATE<br>CHANGE_WIFI_MULTICAST_STATE<br>com.dell.enterpriseservices.SET_PROPERTY_WIFI<br>com.dell.enterpriseservices.SET_PROPERTY_WIFI_PROXY | Disables then re-enables Wi-Fi on the mobile device |
| READ_PHONE_STATE | Iterates through each phone number found inside the application and simulates a call to the phone from those numbers. Alternatively, if no phone numbers were found, it uses a single, random-generated phone number. |

After the permission-specific interactions have run, the script loops over every listed broadcast receiver for the application and sends an instance of that broadcast. It is anticipated that many broadcasts will fail without necessary extra fields; however, some broadcasts may have a result.

Finally, the application is stopped and uninstalled. The dynamic analysis script is finished, and the host process will soon retake control.

### 3.3 Phone

The phone itself does very little except be a phone. The dynamic analysis script installs and launches the application, and it handles all of the interactions.

The dynamic analysis script does alter 1 phone property: "net.dns1". This property is used to identify the phone's domain name server, and it is set to point to the analysis box.

## 4. Results

Over the course of the dynamic analysis process, 3 or 4 output files are generated and stored in the database for eventual in-depth analysis.

The first and optional output file is a screenshot of the application's interface. This output file may or may not exist, depending on whether the application has a graphical interface. Its use in analysis is likely low and would only give an indication of whether the application has a splash screen that might block thorough dynamic analysis.

System calls performed by an application are captured in the strace output file. The strace tool attaches to the application's process ID shortly after launch and captures system calls during the entirety of analysis. After the output file is pulled to the entry box, the host process parses relevant data from each line of the file: process ID, function name, arguments, descriptor, timestamp, and any error codes present. These data are stored in the database under the "a2d.strace" collection and will await further analysis.

Logcat output is handled similarly to strace output. The logcat listener initializes before the application is even installed and constantly listens throughout the dynamic analysis. Once the output file reaches the host process, it parses the date, time, process ID, thread ID, logging level, grouping tag, and the actual log message extracted from each line and stores it in the database under the "a2d.logcat" collection.

Network traffic is captured by a "tcpdump" process running on the mobile device. All traffic would travel across a network with only 2 nodes: the mobile device and its paired analyzer. The IP addresses are statically assigned and consistent for every pair of phone and analyzer (Table 2).

Table 2 IP addresses assigned to VMs

| Virtual Machine | IP Address |
|---|---|
| Mobile device | 192.168.2.3 |
| Analyzer | 192.168.2.2 |

The packet capture (pcap) file is pulled to the analyzer, then to the host process. The host process then iterates over every packet captured and parses potentially useful information for storage in the database, including: the source and destination IP addresses, source and destination ports, timestamp, payload, and potentially other pieces for specific protocols. Each packet is stored individually in the database under the "a2d.pcap" collection.

Eventually, features from these results will be used alongside static analysis results to provide a more accurate threat score, but that is beyond the scope of this paper.

## 5. Conclusions

A2D is currently a static analysis tool only. As a result, it may miss any obfuscated functionality within applications. By expanding into dynamic analysis, better visibility into hidden features should be achieved.

Capturing strace and logcat output can help illuminate malicious functionality hidden in obfuscated or compiled code, whereas captured network traffic can identify malicious servers that a piece of malware may attempt to contact. Activity patterns may also be captured that can differentiate malicious applications from the benign.

Combining these data will give better scoring results and provide analysts additional information for deciding whether a mobile application can be cleared for use on critical devices.

## 6. References

1. pymongo. 2008–2014 [accessed 2014 Dec 18]. http://api.mongodb.org/python/current/.

2. Pointer R, Forcier J. paramiko. 2003–2014 [accessed 2014 Dec 18]. https://github.com/paramiko/paramiko.

3. dpkt. 2013 [accessed 2014 Dec 18]. https://code.google.com/p/dpkt/.

## List of Symbols, Abbreviations, and Acronyms

| | |
|---|---|
| A2D | Application Analysis and Decision (ARL mobile application analysis tool) |
| CPU | Central Processing Unit |
| DNS | domain name system |
| HTML | Hyper Text Markup Language |
| HTTP | Hypertext Transfer Protocol |
| ID | identification |
| IP | Internet Protocol |
| JSON | JavaScript Object Notation |
| OS | Operating System |
| pcap | packet CAPture (a filetype that stores captured network traffic) |
| RAM | Random Access Memory |
| SFTP | Secure File Transfer Protocol |
| SMS | Short Message Service |
| SSH | Secure Shell |
| VM | virtual machine |

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

2 DIRECTOR
(PDF) US ARMY RSRCH LAB
RDRL CIO LL
RDRL IMAL HRA RECORDS MGMT

1 DIRECTOR
(PDF) US ARMY RSRCH LAB
RDRL CIN D
J EDWARDS

INTENTIONALLY LEFT BLANK.