Location-Based Security Framework for Cloud Perimeters

Chetan Jaiswal, Mahesh Nath, and Vijay Kumar,
University of Missouri, Kansas City

A new approach to compose firewall policies to protect mobile and static cloud perimeters uses location to filter out attacks from unsafe locations.

Many enterprise and government organizations use a variety of mobile gadgets to connect to the cloud to manage their data processing requirements. Although this platform improves availability and performance, it also increases security risk, as it can allow unwanted malicious network traffic into the organization. Firewall filtering is often inadequate for stopping these attacks. The problem becomes more complex when multiple firewalls are deployed because coordination among them becomes extremely difficult if not impossible.

Current firewalls use static filtering policies. Although simple, a static policy has many disadvantages. First, because border routers enforce a static policy, they can't react to changes in the external environment. Second, because of physical limitations and differences in trust relationships between an enterprise and its immediate neighbors, some firewalls might require preferential treatment over others in admitting different kinds of traffic streams. Therefore, providing perimeter protection policies that react to dynamic changes and respect organizational objectives such as preferential treatment while enforcing organizations' overall security objectives requires dynamic and flexible policies at each border gateway that are also part of a global policy such that they enforce common security objectives in mobile clouds.

2325-6095/14/$31.00 © 2014 IEEE

The protection issue becomes more complex when we consider attacks from mobile sources. Unlike threats from stationary attackers, mobile attackers disappear from the attack location and resurface elsewhere. We introduce *location-attack protection*, in which the firewall can block messages from high cybercrime locations (country, state, and so on) completely. To enable this protection, we use constrained logic programming by appropriately altering the *flexible authorization framework* (FAF)[1] and its extensions, and strand spaces and multiset rewriting strategies for protocol analysis.[2–4] These two formalisms control multiple streams of data exchanged between two participants, which is relevant to our framework because it requires fine-grained and protocol-specific perimeter protection policies. They can also easily incorporate new spatial and temporal parameters that are unique and crucial to mobile clouds. Our scheme supports consistency and completeness of local policies: they're important because an individual gateway or firewall on the perimeter needs to know whether to allow or deny a stream's progress. The scheme makes sure that the composition of local policies is logically correct to obtain an enterprise-wide perimeter protection policy. In addition, our scheme makes sure that the effect of the propagation of change in policies to others is correct. If the global policy changes, then all local polices have to accommodate that change. Conversely, if a local policy changes, the related global policy may change, which in turn may trigger changes in other dependent local policies.

A mobile unit frequently changes location, which can introduce inconsistency at a policy level. For example, a mobile user might be subject to a different set of constraints in Kansas City, Kansas, than in Kansas City, Missouri, which will affect the data access pattern. Firewall filtering schemes must handle such policy changes, due to mobility and other necessary revisions in the policy in real time to eliminate false denial. This becomes tricky because a mobile unit becomes unreachable when it's switched off or slips into doze mode:[5] updates or changes can only be installed when the unit becomes active. To address this problem, we use a twofold optimization strategy. In the first phase, we apply fold/unfold[6] transformations to optimize policy rules. In the second phase, we partially materialize static parts of individual polices, excluding dynamic variables that share information between local and global policies. We then translate such optimized policies to rule sets used by today's firewalls.[7]

Mobile Cloud

Mobile clouds support personal and terminal mobility.[5] A mobile unit can mount attacks from any location at any time. Attack packets pass through several gateways before reaching the cloud. Each gateway has its own dynamic firewall, and the cloud is protected by its own firewall. Whenever a firewall policy change is incorporated on any of the firewalls, the change is propagated to all other firewalls for updating.

Firewalls are typically configured using a rule base specifying which inbound or outbound packets (or sessions) are to be allowed or blocked. A Cisco rule set[7] is as follows: pass tcp 20.9.17.8 0.0.0.0 121.11.127.20 0.0.0.0 range 23 27, which says that TCP packets from IP address 20.9.17.8 to IP address 121.11.127.20 are to be accepted if the destination port range is from 23 to 27. The 0.0.0.0 segments mean that address masking isn't used. Generally, such rules are listed in some order in access lists.[7] When a firewall receives a packet, it goes through the list and matches the first rule that applies to the packet and follows the specified action. Firewalls use a closed policy that drops packets not explicitly permitted by any rule. This procedure leads to several problems:

- Because the rules are written at the lower protocol level, a misconfiguration can make the whole intranet unreachable.
- The rule base might have many redundant rules.
- The semantics depend not only on the rules, but also on the order in which they're listed, an undesirable feature.

Earlier research on this issue provided solutions with limited success. For example, Yair Bartal and his colleagues proposed Firmato, a firewall management toolkit.[8] Although it models the firewall security policy and network topologies, it doesn't permit fine-grained admission control of streams, doesn't cover intranets with multiple external gateways that enforce different policies, and can't be used to obtain

Rule 1:	procNxtPkt([], post, S_i,+)	←	permToOpen(S_i)
Rule 2:	procNxtPkt(pre*car(post), post, S_i,+)	←	blocked(S_i)
	PROVISION(self):		updtLocalStat(car(post), S_i ,L_i),
			localPktAcpPolicy(car(post), pre, S_i,+),
	PROVISION(global):		gPktAcPolicy(car(post), S_i, [P_1, . . ., P_n],+)
	OBLIGATION(global):		updtGlobalStat(car(post), S_i, [P_1, . . ., P_n],+)
Rule 3:	blocked(x)	←	L_i = maxL_i
(a)			
Rule 4:	procNxtPkt([], post, S_i,+, LO$_i$)	←	permToOpen(S_i, LOi)
Rule 5:	procNxtPkt(pre*car(post), post, S_i,+)	←	blocked(S_i, LOi)
	PROVISION(self):		updtLocalStat(car(post), S_i, L_i),
			localPktAcpPolicy(car(post), pre, S_i,+, LO$_i$),
	PROVISION(global):		gPktAcPolicy(car(post), S_i, [P_1, . . ., P_n],+, LO$_i$)
	OBLIGATION(global):		updtGlobalStat(car(post), S_i, [P_1, . . ., P_n],+,LO$_i$)
Rule 6:	blocked(x)	←	L_i = maxL_i
(b)			

FIGURE 1. Packet acceptance and rejection rules written in Flexible Parameter Protection Specification Language (FPPL) for: (a) local and enterprise-wide policies, and (b) mobile policies.

the global health of the traffic streams entering the Internet. Alain Mayer and his colleagues[9] present Fang, a firewall analysis engine that has the same deficiencies as Firmto.[8] Our scheme resolves these deficiencies. A cryptography-based scheme relies on decentralized trust management.[10] Our solution distributes network perimeter protection without relinquishing centralized control and thereby circumvents the performance bottlenecks of a centralized perimeter protection policy.

Unlike wired systems, a mobile node can issue a request from any location, connect to many service providers that might have different security requirements, slip into doze mode, power off, or fail. Mobile nodes are also vulnerable to attacks. A mobile client's valid request from one location can be denied at another location. Several good schemes have been proposed for protecting mobile systems through firewalls[11]; however, they provide engineering solutions to firewall protection and appear highly system dependent.

We conclude the following: we need real-time synchronization of firewalls and subsequent updates; a multilayer verification is the way to go; and the system must implement geographical location-based verification. Our logic-based framework meets these requirements.

Flexible Perimeter Protection Framework

Because a common framework can protect mobile and wired traffic, we built a unified flexible framework to monitor and dynamically adjust the entering datastreams. FPPF is based on having rules built with predicates to express policies for accepting packets in an ongoing stream. We wrote the FPPF filter or protection rules in the *Flexible Parameter Protection Specification Language* (FPPL). We briefly introduce salient features of this language here; details can be found elsewhere.[1]

Example Policies Written in FPPL

FPPL, similar to other logical languages, consists of constant symbols, variables, function symbols, and terms. It uses a set of predicates to define packet acceptance and rejection rules for local and enterprise-wide policies.

For example, the rules in Figure 1a define a local policy. Rule 1 says that stream S_i can be opened if it has been permitted to do so, where permToOpen(S_i) holds when the latter is true. Rule 2 says that the next packet of S_i is admitted as long as S_i isn't blocked, the local packet acceptance and the global approval policies allow it, and the corresponding local and global statistics are updated. Rule 3 defines the condition for S_i being blocked—namely, that the local variable L_i (say buffer capacity allocated to this stream) has been used up by the stream up to now.

The local policy needs to know that the predicate gPktAcPolicy(car(post), S_i, [P_1, . . ., P_n],+) (a part of enterprise-wide policy) is true for the next packet to be admitted according to the agreement it has with the enterprise-wide security policy. Conversely, it's obligated to update the global statistics updtGlobalStat(car(post), S_i, [P_1, . . ., P_n],+) that are a part of the enterprise-wide security policy. Note that other variables appearing in these two predicates, namely [P_1, . . ., P_n], are unknown to the local policy, so they can't be used in the rule as a normal predicate. The enterprise-wide policies are composed accordingly.

Enhancing Provisions, Obligations, and Delegations

Provisions and obligations play a key role in the FPPF architecture. As the previous example demonstrates, local policies depend on having provisions approved by the global policy base, and, in turn, local policies are obliged to update their local statistics with the global policy base. This two-way exchange of data allows the global policy to respond to perimeter-wide changes in an accurate and timely manner.

In our case, the provision granted by the global policy base to the local policy is specified in the predicate gPktAcPolicy(car(post), S_i, [P_1, . . ., P_n],+). Therefore, we model a provision as a predicate exported by the grantor and imported by the grantee. The main characteristic of the provision is that the grantee doesn't know its definition, but would know if it's evaluated to be true or false.

The obligation used in the example is the predicate updtGlobalStat(car(post), S_i, [P_1, . . ., P_n],+). Note that this is also a predicate that's exported to the local policy base, which must instantiate the proper instances of variables that would make the predicate instance true. This obligation can be fulfilled when the function call is made.

Policy Updates in Mobile Systems

Geographical location plays an important role in managing mobile activities (such as an attack). We include geographical locations in the predicates used to specify local and enterprise-wide policies. The firewall policy in the cloud will depend on where S_i originates (location-specific attacks). Thus, the predicates for the local policy will include attacker's location, and the predicates for the enterprise-wide policy will depend on a set of locations where a mobile unit is permitted to roam.

We illustrate mobile policy composition with a modified rule and example (Figure 1b).

In this example, rule 4 says that S_i can be opened if it originated at location LO$_i$ (longitude), and if it has been permitted to do so, where permToOpen(S_i) holds when the latter is true. Rule 5 says that the next packet of S_i is admitted provided that S_i isn't blocked, local packet acceptance and the global approval policies allow it, and the corresponding local and global statistics are updated. Rule 6 defines the condition for S_i being blocked—namely, that the local variable L_i (say, buffer capacity allocated to this stream) has been used up by the stream up to now.

In mobile attacks, the attack location can change frequently. As a result, firewall policies can change frequently, leading to increased update traffic, which might not be able to handle such frequent updates and might not be able to keep the local and global policies in sync. Our scheme addresses this issue by keeping the policy warehouse at all base stations. Because a base station serves a specific location, policy relevant to that location is loaded there. A mobile unit will cache the policy from the base station of the cell it's visiting. A base station will broadcast policy changes as they occur, and all mobile units in that cell will capture it and visitors to that cell will acquire it when they register.

Unsafe Locations

In our experience, more attacks (serious or less serious) come from some locations than others. The predicates coded in the firewalls in our system include a location parameter to identify an attack's origin. If it originates from an "unsafe" location, it's blocked. We define three categories of unsafe locations.

A *hard* location is one from which numerous serious attacks originate with high frequency. Any of these attacks can severely affect the cloud's performance and integrity. The firewall must stop these attacks. If the firewall detects that an attack (for example, a Trojan) is mounted from a serious location, it immediately eliminates this attack.

A *soft* location is one from which relatively fewer serious attacks originate. These types of attacks don't significantly affect the cloud's performance and integrity, and the cloud system can continue to function while the firewall handles the attack. For example, a music sharing virus can scare people without harming the computer. The firewall might let it enter the cloud.

Finally, a *clean* location is one from which no attacks originate. The firewall might apply minimal security checks to messages coming from these locations.

Unsafe Location Identification in Mobile Communication

When an attacker moves around in a location while attacking the cloud, it will have the same IP address at different points inside the location. For example, if an attacker moves from point l_i to point l_j inside a location L, the IP address will not change, that is, points l_i and l_j will have the same IP address even though their geographical address (point l_i to point l_j) inside L will change. Thus, to hide their identity and avoid being caught in such movement, attackers

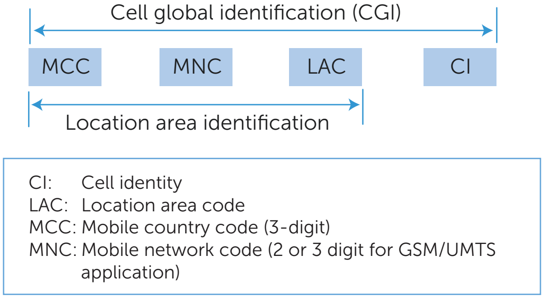
Cell global identification (CGI)
MCC
MNC
LAC
CI
Location area identification
CI: Cell identity
LAC: Location area code
MCC: Mobile country code (3-digit)
MNC: Mobile network code (2 or 3 digit for GSM/UMTS application)

FIGURE 2. Cell global identitification.

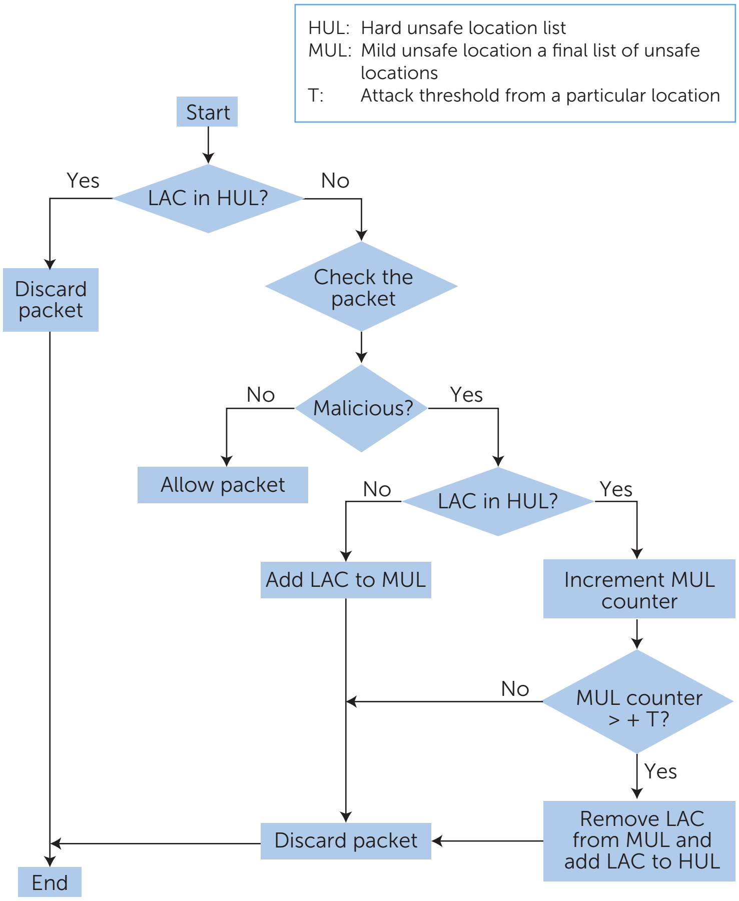
HUL: Hard unsafe location list
MUL: Mild unsafe location a final list of unsafe locations
T: Attack threshold from a particular location
Start
LAC in HUL?
Yes
No
Discard packet
Check the packet
Malicious?
No
Yes
Allow packet
LAC in HUL?
No
Yes
Add LAC to MUL
Increment MUL counter
MUL counter > + T?
No
Yes
Remove LAC from MUL and add LAC to HUL
Discard packet
End

FIGURE 3. Determining unsafe locations.

generally use a proxy. The *tracert* (trace route) command, which shows the path an IP packet travelled to reach a destination, isn't helpful because it can't go beyond the proxy.

In a mobile network, IP address allocation is dynamic, so it's easy for an attacker to spoof an IP address and mount an attack through a proxy. To reach the attacker behind the proxy, our approach identifies a mobile phone's location in a cellular network through its *cell global identity* (Figure 2).

At present, cell global identity information is not available in IP packets coming from a mobile device. Our scheme (Figure 2) extends the structure of an IP address and includes cell global identity information in IP packets. This helps us to identify the location of the mobile unit mounting the attack, directly or through a proxy, and to program the firewall accordingly to block the attack. For example, if the cell global identity in an IP packet is mobile country code (MCC) = 310 (indicates USA) and mobile network code (MNC) = 410 (AT&T network), location area code (LAC) = 3450 and cell identity (CI) = 118541125 represents a cell in Kansas City, Missouri.

As a security measure, the system maintains location area codes of unsafe locations and discards incoming packets from these unsafe locations without even analyzing them. To determine if a location is safe or unsafe, the system records the number of attacks from each known location. If the number of attacks from a particular location reaches a threshold, it marks the location as unsafe. Figure 3 illustrates our approach for determining unsafe locations. It's an ongoing process of finding unsafe locations based on the number of attacks originating from a specific location. We maintain a database of unsafe locations.

The *serving GPRS support node* (SGSN) is the main component of the General Packet Radio Service network. The SGSN can make this location information available in IP packets coming from mobile stations because it has access to the location information (CGI) of a mobile station in its area and is also responsible for delivering data packets to and from the mobile stations in its area. The IP packet header, which contains an optional field, can be used to store the cell global identity.

Figure 4 shows the flow of IP packets in the Global System for Mobile Communication (GSM) and Universal Mobile Telecommunications System (UMTS) architectures. We've included relevant elements of GSM in our scheme for identifying unsafe locations. On the receiving end, the firewall extracts the CGI information from the enhanced IP packet and searches hard unsafe location (HUL) and mild unsafe location (MUL) lists and decides whether to reject or allow the packet.

Because our approach can identify the attacker's location, it compromises the attacker's privacy. Although this is not an issue in the case of an attacker, our scheme should be able to protect the privacy (which could lead to a security breach) of a typical user if that user's actions look like an attack. We're investigating a solution to make sure that the loca-

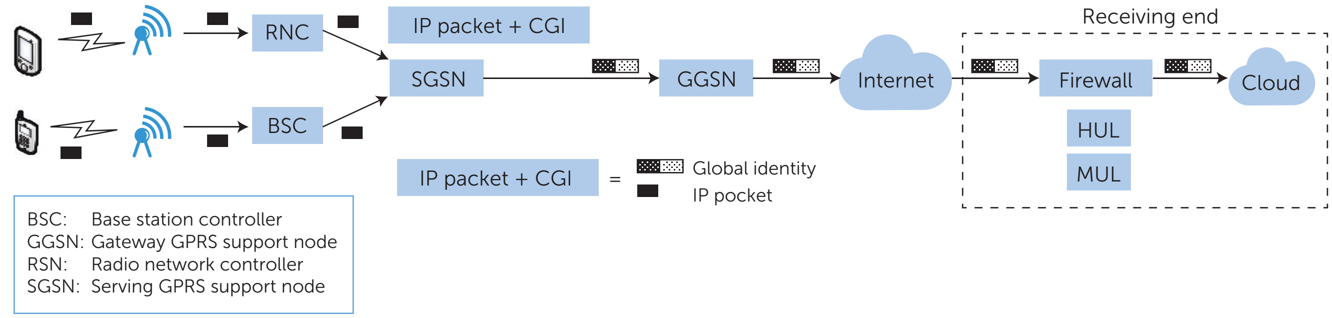
RNC
BSC
IP packet + CGI
SGSN
GGSN
Internet
Receiving end
Firewall
Cloud
HUL
MUL
IP packet + CGI =
Global identity
IP pocket
BSC: Base station controller
GGSN: Gateway GPRS support node
RSN: Radio network controller
SGSN: Serving GPRS support node

FIGURE 4. Partial network architecture: GSM + GPRS + UMTS.

tion isn't accessible to anybody or only its encrypted version is accessible.

Unsafe Locations Identification in Wired Networks

Because our scheme for identifying unsafe locations in a cellular platform won't work for stationary attackers, we developed a different solution for these attackers. We use known *landmarks* (that is, trusted computers on the Internet whose geographic coordinates are known a priori) to probe an attacker's machine (at an unknown location) and measure the response delay to compute attacker coordinates. We repeat this process until we identify the location as accurately as possible.

The packet transfer delay is directly proportional to the distance. Line congestion, queuing delays, and so on can affect this relationship, but by consolidating several observations, we can identify the location with reasonable accuracy. We probe the destination from several landmarks to get a delay vector, which we then use to get an overlap area as the destination. When we have many landmarks, we can triangulate the results to determine the attack's geoposition. In our approach, we first create a large dataset of real-world measurements by measuring latency from each landmark to every other landmark. We used PlanetLab (www.planet-lab.org) and a carefully selected geographically diverse set of landmarks across the globe. We used triangulation to implement our algorithm, which uses three out of 50 landmarks as the starting point. It iterates with three different landmarks until we obtain consistent results. Our observation concurs with the other work.[12]

Our algorithm (see the sidebar) starts with a set of continental landmarks (CLMs) measuring the delay in reaching the attacker machine (AM$_X$). We considered the landmark at the University of California, Berkeley, as the west CLM (CLM$_2$) and the landmark at Michigan Technological University as the north CLM (CLM$_3$). The algorithm first randomly selects three landmarks, each from a different CLM dataset (CLM$_1$, CLM$_2$, CLM$_3$, or CLM$_4$). In the next step, each LM-CLM$_i$ (where $1 \leq i \leq 3$) individually measures the delay to AM$_X$. Using the AvgLDelay$_i$ (distance to delay measurements based on average lowest delay between any two landmarks), each LM-CLM$_i$ estimates the distance to AM$_X$.

We create AvgLDelay for each LM-CLM, which provides the average ratio of distance to delay (DDR). After estimating the distance of AM$_X$ from the three LM-CLMs, our algorithm ascertains the geolocation (AM$_{LA}$, AM$_{LO}$) of AM$_X$. In step 5, the algorithm considers the area (Zonal_Region) surrounding (AM$_{LA}$, AM$_{LO}$), called the *initial zone*. After identifying the initial zone, it creates the AvgLowestNodetoZoneDelay (dataset of distance-to-delay measurements based on the average lowest delay between a particular node to the zone with AM as the given latitude and longitude) for each selected LM-CLM$_i$ on the fly. In the next step, each LM-CLM$_i$ individually measures the delay to AM$_X$. Using AvgLowestNodetoZoneDelay$_i$, each LM-CLM$_i$ estimates the distance to AM$_X$. After estimating the distance of AM$_X$ from the three LM-CLMs, the algorithm ascertains the new geolocation (AM$_{LA1}$, AM$_{LO1}$) of AM$_X$. In step 8, it finds the set of zonal landmarks (ZLMs) in the zonal region (initial value ±4°) around (AM$_{LA1}$, AM$_{LO1}$), which we call the *final zone*.

It's important to find landmarks that are diverse with respect to each other as well as to (AM$_{LA1}$, AM$_{LO1}$). Once the final zone is identified, the algorithm creates AvgLowestZonalDelay on the fly by considering the prerecorded minimum delays from each LM-ZLM$_i$ to LM-ZLM$_j$ ($\forall i, j$: $i \neq j$, $1 \leq i \leq n$, $1 \leq j \leq n$, each LM-ZLM $\in$ Final_Zone where n is the total number of landmarks in the final zone). This dataset provides the final zone's DDR. In the next step, each LM-ZLM$_i$ measures the delay to AM$_X$.

ALGORITHM STEPS

Here, we describe the steps in our algorithm. We consider two cases: the general case, in which more than three landmarks are required to ascertain the geolocation; and the best case, in which only three landmarks are required to ascertain the geolocation.

Total number of landmarks (LMs) = N

1. Select any three LMs (LM-CLM$_1$, LM-CLM$_2$, LM-CLM$_3$) from the CLM sets (Figure A).
2. Calculate AverageLDelay at each LM-CLM$_i$ to attacker machine AM$_X$.
3. Estimate distance (CLM-Dist$_i$) from each LM-CLM$_i$ to AM$_X$ based on AvgLDelay.
4. Ascertain the location of AM$_X$ using trilateration as (AM$_{LA}$, AM$_{LO}$).

We used the great circle and aviation formulas[1] to perform our calculations. We know the length of the sides of triangle ABC (Figure A1), which is the distance between the known landmarks. Using the lengths (AB, BC, CD), angles, and the estimated lengths of AD, BD and CD, we apply the triangulation to get the coordinate of point D (AM$_{LA}$, AM$_{LO}$).

5. Find all LMs in ±4º (Zonal_Region) of (AM$_{LA}$, AM$_{LO}$). We refer to this as the Initial_Zone.

We refer to "find all the LMs in ±4º (Zonal_Region) of (AM$_{LA}$, AM$_{LO}$)" as Initial_Zone. Because we know the geolocations of all landmarks, our algorithm finds the landmarks in the Initial_Zone (Figure A2).

6. Create AvgLowestNodeToZoneDelay.
7. Estimate distance (CLM-Dist$_i$) again from each LM-CLM$_i$ to AM$_X$ based on AvgLowestNodeToZoneDelay.
8. Ascertain AMX location using triangulation as (AM$_{LA1}$, AM$_{LO1}$).

The algorithm calculates the new geolocation of AM$_X$ based on current zonal DDR (Figure A3). D$_1$ is the new geolocation of AM (AM$_{LA1}$, AM$_{LO1}$).

9. Find all LMs in the Zonal_Region of (AM$_{LA1}$, AM$_{LO1}$). Call this Final_Zone if Total_LMs in Zonal_Region < 3. Then exit with result as (AM$_{LA1}$, AM$_{LO1}$).
10. Select any three LMs (LM-ZLM$_1$, LM-ZLM$_2$, LM-ZLM$_3$) from Final_Zone based on the top values of Diverse$_j$ = ABS (LM-ZLM$_j^{LA}$ − AM$_{LA1}$, LM-ZLM$_j^{LO}$ − AM$_{LO1}$) and ABS(LM-ZLM$_j^{LA}$ − LM-ZLM$_i^{LA}$, LM-ZLM$_j^{LO}$ − LM-ZLM$_i^{LO}$).

Figure A4 shows the zonal landmarks (ZLMs) of the Final_Zone. These are the LMs in the Zonal_Region of (AM$_{LA1}$, AM$_{LO1}$). Because we only need three LMs in each iteration, our algorithm computes the diversity parameter for all the LMs in the zone and, based on this parameter, selects the three LMs with the highest values (Figure A5).

11. Create AvgLowestZonalDelay.
12. Calculate AverageOfLowest delay at each LM-ZLM$_i$ to AM$_X$ and estimate distance from each LM-ZLM$_i$ to AM$_X$ based on Dataset _AvgLowestZonalDelay.
13. Ascertain AMX location using triangulation as (AMLA$_2$, AMLO$_2$) (Figure A6).

After step 13, there are two locations of AM$_X$ as (AM$_{LA1}$, AM$_{LO1}$) and (AM$_{LA2}$, AM$_{LO2}$).

14. Set Zonal_Region = Zonal_Region − Closing _Factor.
15. Compare (AMLA$_1$, AMLO$_1$) and (AMLA$_2$, AMLO$_2$). If the result = satisfactory or Zonal_Region = 0º, then exit with result as (AMLA$_2$, AMLO$_2$).

If the result is satisfactory in the first iteration, the algorithm terminates. Thus, steps 9 through 15 validate the results. This is the best case because only three LMs can identify the AMX location accurately in just one iteration. Because the result is unsatisfactory, the algorithm iterates steps 9 through 16, as follows:

16. Goto step 9 with (AM$_{LA1}$, AM$_{LO1}$) = (AM$_{LA2}$, AM$_{LO2}$).

Reference

17. E. Williams, *Aviation Formulary V1.46*, http://williams.best.vwh.net/avform.htm.

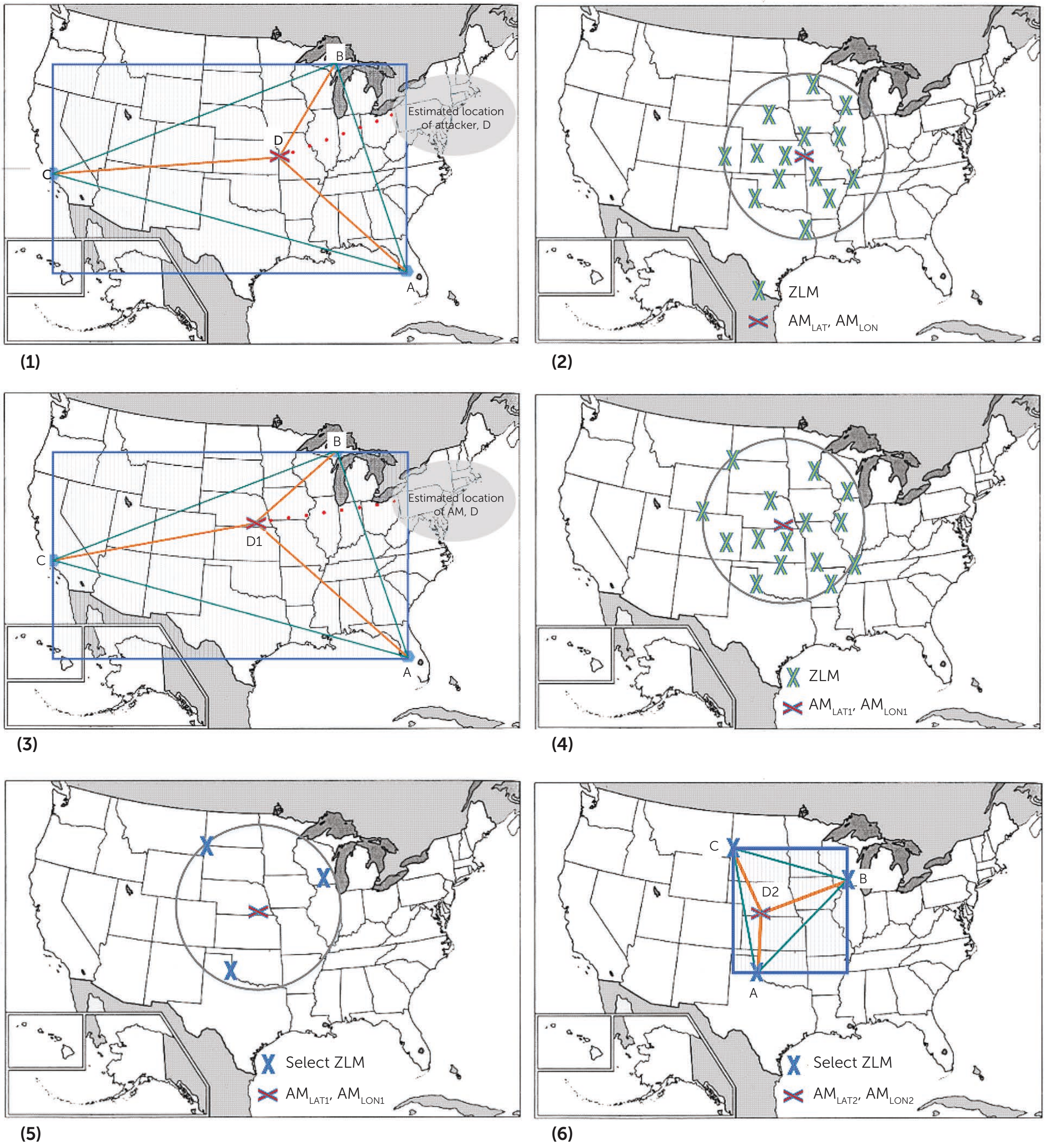

FIGURE A. Landmark selection and attacker location estimation: (1) calculate lengths of the sides of triangle ABC; (2) find the landmarks in the Initial_Zone; (3) calculate the new positions of the attacker machine AMX; (4) determine zonal landmarks (ZLMs) of the Final_Zone; (5) select the three landmarks (LMs) with the highest values; and (6) ascertain the attacker machine location using triangulation.

By using AvgLowestZonalDelay$_i$ each LM-ZLM$_i$ estimates the distance to AM$_X$. After estimating the distance of AM$_X$ from the three LM-ZLMs, the algorithm ascertains the new geolocation (AM$_{LA2}$, AM$_{LO2}$) of AM$_X$. It then compares the two geolocations (AM$_{LA2}$, AM$_{LO2}$) and (AM$_{LA1}$, AM$_{LO1}$) for error distance.

If the error distance is less than 10 miles, we consider the result satisfactory, and the algorithm terminates with (AM$_{LA2}$, AM$_{LO2}$) as the final geolocation of AM$_X$. It can also terminate when the zonal region reaches zero or the total number of landmarks in the Zonal_Region is less than 3. In other cases, the algorithm continues to iterate until it reaches the desired location accuracy.

Our current work will provide us with a platform for dealing with the security of global cloud structure (linking all customers who rent cloud services). At present, banks are reluctant to use cloud services because they don't know the whereabouts of datacenters. Our system will identify a datacenter's geographical location and dynamically manage the firewall protecting it. This approach will largely relieve cloud service providers from the responsibility of securing their datacenters.

Acknowledgments

We thank Sushil Jajodia for his highly useful suggestions, which helped us improve the algorithm for firewall composition. US National Science Foundation grant *CNS-1347958* supported this research.

References

1. S. Jajodia et al., "Flexible Support for Multiple Access Control Policies," *ACM Trans. Database Systems* (TODS), vol. 26, no. 2, 2001, pp. 214–260.
2. I. Cervesato et al., "Relating Strands and Multiset Rewriting for Security Protocol Analysis," *Proc. 13th Computer Security Foundations Workshop* (PCSFW 00), 2000, pp. 35–51.
3. F.J. Fabrega, J.C. Herzog, and J. Guttman, "Strand Spaces: Why Is a Security Protocol Correct?" *Proc. IEEE Symp. Security and Privacy*, 1998, pp. 160–171.
4. J. Loeckx and K. Sieber, *The Foundations of Program Verification*, John Wiley & Sons, 1987.
5. V. Kumar, *Mobile Database Systems*, John Wiley & Sons, 2006.
6. H. Seki, "Unfold/Fold Transformation of Stratified Programs," *Theoretical Computer Science*, vol. 86, no. 1, 1991, pp. 107–139.
7. *Cisco ISO Lock and Key Security*, white paper, Cisco Systems, 1996.
8. Y. Bartal et al., "Firmato: A Novel Firewall Management Toolkit," *Proc. IEEE Symp. Security and Privacy*, 1999, pp. 17–31.
9. A. Mayer, A. Wool, and E. Ziskind, "Fang: A Firewall Analysis Engine," *Proc. IEEE Symp. Security and Privacy*, 2000, pp. 177–187.
10. S. Ioannidis et al., "Implementing a Distributed Firewall," *Proc. ACM Conf. Computer and Comm. Security*, 2000, pp. 190–199.
11. E. Goren and O. Duskin, "Mobile Firewall," internal report, Check Point Software Technologies, Hebrew Univ.
12. M. Gondree and Z.N.J. Peterson. "Geolocation of Data in the Cloud," *Proc. 3rd ACM Conf. Data and Application Security and Privacy*, 2013.

CHETAN JAISWAL *is a PhD scholar at the University of Missouri, Kansas City. His research interests include cloud computing; mobile, wireless sensor networks, and cloud security; and cloud-based database transaction systems. He is also passionate about programming, learning new concepts, and teaching. Contact him at chetanjaiswal@mail.umkc.edu.*

MAHESH NATH *is a PhD scholar at the University of Missouri, Kansas City. His research interest include network and information security and privacy, with an emphasis in next-generation firewall frameworks. Contact him at mnmg2d@mail.umkc.edu.*

VIJAY KUMAR *is the Curator's Professor in the computer science department at the University of Missouri, Kansas City. His research interests include information security, wireless and mobile computing, and database systems, with particular emphasis in cybersecurity and wireless data dissemination. Kumar has a PhD in computer science from Southampton University, England. Contact him at kumarv@umkc.edu.*