

Pushing the Security Boundaries of Ubiquitous Computing

David Llewellyn-Jones, Madjid Merabti, Qi Shi, and Bob Askwith

School of Computing and Mathematical Sciences, Liverpool John Moores University*
{d.llewellyn-jones, m.merabti, q.shi, r.askwith}@ljmu.ac.uk

Abstract. As we move towards Ubiquitous Computing environments, the nature of the networks that are used for device communication is changing: becoming more fluid and with blurred boundaries that are no longer fixed. At present many security techniques rely on the incumbent perimeter model that assumes relatively fixed boundaries. As a compromise between the two, we present an algorithm for establishing boundaries that can be applied to dynamic environments where the network structure is constantly changing. Such a technique can be used to apply security policies in dynamic environments where boundaries are fluid, but such boundaries may nonetheless be important for the application of the policy. We describe the algorithm and present an analysis and experimental results to show the complexity of the algorithm.

1 Introduction

The natural characteristics and goals of Ubiquitous Computing environments run contrary to many security requirements, resulting in considerable challenges for the development of security techniques. The heterogeneous and dynamic nature, with fluid data and code movement across ubiquitous and wireless networks, context awareness and low-power devices all combine to make security onerous [1]. Central to the potential benefits of Ubiquitous Computing – and of particular concern from a security perspective – is the dissolution of barriers and boundaries that currently pervade network space. Using existing security tools, these provide a relatively effective weapon against a number of network-borne threats. For example, in many organisations firewalls, gateways and proxy servers provide safety from external threats such as viruses, spyware, denial of service attacks, network intrusion and the leakage of confidential data [2]. Whilst the use of perimeter defences for security is perhaps imperfect in that it can leave an organisation vulnerable to internal attacks and can restrict the utility of the network, nonetheless their widespread use (the 2005 CSI Computer Crime and Security Survey showed that 97 % of responding organisations used firewalls [3]) suggests that boundaries remain an important element of network security defences.

* This work has been made possible by a UK EPSRC grant (reference GR/S01634/01) under the Programmable Networks initiative.

If boundaries provide an important part of network defences, the question therefore arises as to how security can be maintained in a Ubiquitous Computing environment.

Work in distributed and mobile security (distributed trust [4], distributed intrusion detection systems [5], mobile firewalls [6] *etc.*) may go some way towards allowing seamless Ubiquitous Computing operation without the usual defensive boundaries. However, there is clearly some way to go before such techniques become fully accepted, and the answer to the question of how security can therefore be maintained is not straightforward.

In this paper, we will present work that we believe may lead to an effective middle ground between the current segregated network structure and future boundary-free interoperation.

Central to the idea we present is that if boundaries are to exist, they should at least be dynamic and flexible. For example, if a device enters a network, it should automatically work within the prescribed boundaries, setting itself up to fit the policy requirement within the boundaries, and allowing assurance that it will not itself ‘breach’ the boundaries (even unwittingly) by bypassing firewalls, for example. At present, it is largely down to the configuration of individual devices as to whether they work within the boundary, outside it, or even straddle it [7]. Yet the security implications affect the entire network. We therefore present a method for analysing network topologies to ascertain those nodes that are working inside, at, or beyond the boundaries, so that they can be effectively distinguished from the perspective of network administrative systems. Additional security can then be imposed on boundary nodes to ensure the security integrity of the network is preserved.

Since the procedure presented is simple and does not require significant resources, it is intended that the boundary can be maintained dynamically, with security being updated as the network changes and devices enter or leave it.

This work focusses primarily on the analysis algorithm used to identify boundaries and is not intended as a complete security solution. We nonetheless believe the process provides a novel way of analysing network boundaries and by building the algorithm using a generic composition tool, we have left the possibility open for us to develop it further and test it in a real Networked Appliance environment in the near future. At present, little work has been done dealing with the effect of large-scale composition of systems in ubiquitous computing environments. Perhaps the closest methods to those described here relate to autonomous computing, self-configuring security methods and design patterns [8, 9]. In contrast to these, we present a very simple and robust example of a composition technique that can be used in conjunction with existing border security methods.

This paper is structured as follows. The next section briefly describes the algorithm and discusses some security applications in Ubiquitous Computing environments. We go on to describe the algorithm in detail and our implementation of it in Section 3. Section 4 presents timing results based on our implementation. We conclude and discuss future work in Section 5.

2 Boundaries in a Ubiquitous Computing Environment

There are a number of reasons why it might be useful to understand network topology – and especially boundary structure – in a Ubiquitous Computing environment. In this section we discuss two particular cases: gateway configuration and Composable Assurance analysis.

2.1 Gateway Configuration

Boundary security in a network is often maintained through what we might reasonably describe as ‘Gateways.’ We use the term in the broadest sense, to refer to a device through which network traffic between the inside and outside of a network is directed. Examples of gateway security services might include Intrusion Detection Systems, remote firewalls, proxy web cache/filters, email servers with spam filtering and Network Address Translation devices. The gateway provides a barrier between the inside and outside of the network. However, gateways invariably need configuring when a device enters a network, and a misconfigured device can lead to the rest of the network becoming vulnerable. Devices with multiple network connection capabilities are of particular concern. For example, if a virus arrives through one non-protected connection on a device running on two separate networks, the virus may then be able to propagate to other machines within the firewalled boundary of the second network. This is a perfect example of where boundary detection may be useful: by realising that a device is ‘straddling’ two separate networks in this way, a potential security loophole can be eliminated. In the real world, this might take the form of a device connected to a corporate network whilst also being connected to an additional wireless or dial-up network.

Ultimately, it may be unwise to leave the security of a network to the correct set up of individual device configurations. A better solution would be to allow services running in the network to establish the boundaries of the network and then require devices within the boundaries to reconfigure their network settings in order to participate in the network. In order to do this, such services must first understand where the network boundaries lie and establish a global picture of the internal network in order to interpret interactions between devices.

Although applicable in any network, there are particular benefits to using such a method in a Ubiquitous Computing environment. First, it supports device mobility and dynamic network changes, since it allows detection and reconfiguration as devices enter and leave the network in real time. Second, it supports low-resource devices, since in effect it allows such devices entering a network to out-source security processes to services already running as part of the network infrastructure, if they exist. For example, as a mobile device enters a new network, it is sensible for infrastructure services to take over the management of firewall and network virus protection checking. The alternative is to have such security services running locally on each individual mobile device. This is clearly inappropriate when devices are lower power and resident in a network where more powerful and distributed services are available.

2.2 Composable Assurance Analysis

Secure Component Composition provides a means of assessing the security of composed systems based on the properties of the individual components and the way in which they interact with each other. A result of Shi and Zhang [10] showed the existence of a generalised composition property, known as Composable Assurance, that provides weakened requirements for the application of other composition results, without compromising the security of the composed system.

For example, Shi and Zhang prove that components connected in a directed tree with external interfaces individually satisfying a noninterference property, together form a system that also satisfies the noninterference property as long as all non-external arrival and departure interfaces satisfy a weaker Composable Assurance property. The crucial point is that the weaker components do not compromise the security of the composed system.

The use of these properties is demonstrated by analysing a hypothetical file system that must satisfy a noninterference requirement.

We note that such requirements can be imposed on components of a system as they exist in a networked environment. However, in order to apply the Composable Assurance properties, it is first necessary to distinguish between the various interface types (external, arrival and departure) of the composed nodes. The algorithm we present provides a method for achieving this in a distributed system through an automated technique.

A way to tackle this would be to have the analysis performed by a service within the network, which can then consider whether Composable Assurance and noninterference properties are satisfied on the interfaces of the components, based on the result of the given algorithm. We present only the method for establishing interface types in an automated way, leaving the analysis of the additional properties as future work. However, we claim that this provides an important initial framework for the application of such composition properties in real-world networked systems.

3 Establishing Boundaries Through Analysis

3.1 Overview

The implementation and testing of the analysis process was completed using the MATTS component composition testing tools. The MATTS framework constitutes a series of tools developed for the purpose of designing and testing composition analysis techniques.

The composition analysis techniques created in this way are built up using a combination of certification, formal analysis and topology analysis. From the perspective of the work presented here, the topology analysis is the most important. This topology analysis utilises an XML scripting technique, in which algorithms can be encapsulated. A dual pointer and single stack are used with one pointer relating to the current position in the script and the other relating to the current position in the topology. Each element of the XML script determines

Fig. 1. Boundary analysis script

```
1. <!DOCTYPE compose PUBLIC "http://www.cms.livjm.ac.uk/PUCsec/dtds/compose.dtd"
2. "compose.dtd">
3. <compose type="extended">
4.
5. <sandbox id="s3" config="c1">Boundary check</sandbox>
6. <property id="idExt">External component</property>
7. <property id="idEnc">Encrypted link</property>
8.
9. <configuration id="c1" init="1">
10. <component>
11. <process id="safe" init="1" action="safe=1"/>
12. <process init="0" cond="@a[@n][idExt]" action="safe=2" config="check"/>
13. <input format="" cond="safe&gt;=1"/>
14. <input format="*" cond="safe&gt;=1"/>
15. <output format="*" config="c1" cond="safe=1"/>
16. <output format="*" follow="no" cond="safe=2"/>
17. <output format=""/>
18. </component>
19. </configuration>
20.
21. <process id="check">
22. <process action="link=@ilnum[@n]"/>
23. <process id="link" init="0">
24. <process action="link=(link-1)"/>
25. <process init="0" cond="(!@a[@iln[@n][link]][idExt]) &amp;&amp;
26. (!@ao[@iln[@n][link]][@ilol[@n][link]][idEnc])" action="safe=0"/>
27. <process cond="link &gt; 0" config="link"/>
28. </process>
29. </process>
30.
31. </compose>
```

how the two pointers can be updated at each stage, with movements stored on the stack. If the pointer movement determined by the current element of the script is consistent with the actual topology being tested, and this holds for the entirety of the script, then the topology is considered to satisfy the script. A failure will cause backtracking until ultimately the topology fails to satisfy the script.

In our case, the algorithm and associated script are relatively straightforward. The complete script can be seen in Figure 1. We will now go on to explain the details of this script.

3.2 The Analysis Script

The script is split into two important sections. Lines 9–19 traverse the components that the system is comprised of. Lines 21–29 undertake the actual testing process. Properties are pre-assigned to components and links between components stipulating whether a component is ‘external’ and whether the link is encrypted. These properties are used for demonstration purposes, and could be changed to suit an alternative situation as needed. We will discuss how these properties might be established presently.

The algorithm is interested in relationships between external components and those that are connected to them. Lines 1–8 set up the script, define the purpose of the script and specify the component properties that the script needs to know about. Lines 9–19 then produce a negotiation of the component structure using a depth-first traversal – the standard traversal used by MATTS – and for each component tests whether it is external to the network (line 12). If it is, it then performs a more detailed check of each link entering this component (lines 22–28). For each such link, a check is undertaken to establish whether it is simultaneously from a non-external component whilst also sending unencrypted data (lines 25–26). Should this be the case, the script aborts with a failure result.

The check as to whether a node is external is also used to determine the boundaries of the traversal. We assume that the network of internal nodes forms a connected space. If the node is internal, we therefore continue on to any nodes attached to it (line 15), whereas if the node is external deeper nodes attached to it are not checked, since they will either be external themselves, or attached to the internal network by an alternative route (line 16). Note that MATTS automatically ensures that nodes are visited only once, preventing loops in the network from causing infinite recursion in the algorithm.

The resulting script succeeds only if every link from an internal to an external node is encrypted. If this is not the case, the script will fail, indicating a potential security problem in the network. It achieves this by recognising precisely which components communicate from any internal location to an external node. The script could be easily adjusted for other circumstances, for example ensuring that all external links emanate only from pre-specified proxy servers.

The script deals only with the topological structure of the network. It is also necessary to consider how properties of individual components are established. In the case described, we are interested primarily in whether a node is external, and whether a link is encrypted. Note that the question of whether links between internal nodes are encrypted is of no interest to the script. Only links that cross the boundary from inside out are of relevance.

In general, the details about how to establish such properties will depend on the circumstances and properties in question. However, we note that on IP networks, whether a node is internal or external can generally be established through examination of the IP addresses associated with a node.

For the encryption status of a link, we are currently using properties assigned to links through the equivalent of certification. In this scenario, certification is used to verify that certain output channels from a component send only encrypted data. This certification may be added by the component producer or vendor, or may be verified in advance in some other way. For a discussion of what should happen if the script identifies a problem, or if a component has not been certified, see the next section on resolving failures.

In general, requirements are likely to relate to the security policy in force on a particular network. This might stipulate, for example, a requirement to only connect to external machines using SSL/TLS, in which case a means of identifying the use of such connections would be required. In the case of a proxy

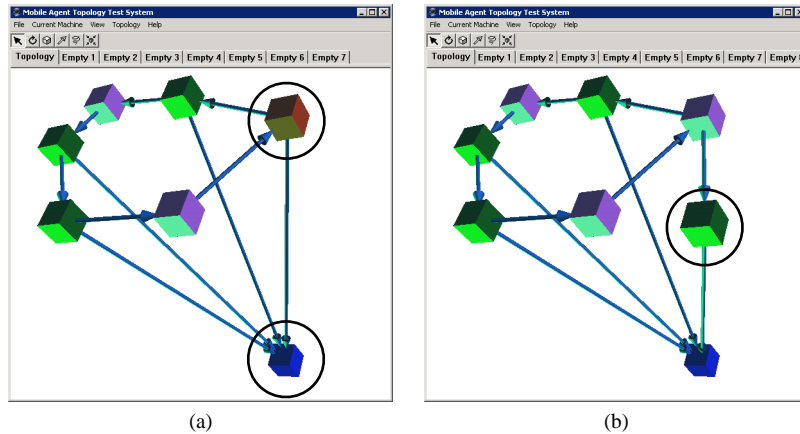


Fig. 2. The identified problem (a) and solution component (b).

server, the details of the server would be unlikely to change often, and so could be stipulated in advance.

3.3 Resolving Failures

An important question that arises is how to deal with the cases where the script fails, indicating a breach of the security policy for the network.

The obvious solution in this case is simply to prevent any non-compliant devices from accessing the network. However, this breaks the Ubiquitous Computing requirement of seamless operation, since it would leave the device in question without a connection.

The ideal for a system that uses component composition for security analysis is to not only highlight problems where they occur, but also to provide solutions that may allow the systems to carry on operating, potentially with slightly less functionality, but with an equivalent level of security.

In the case of encrypted channels described above, the script not only provides warnings when things go wrong, it also provides a means for generating solutions. A failed script will identify the non-compliant device, and the external device that it is attempting to generate a connection with, as can be seen in Figure 2a. In this case the security service could therefore generate an additional encryption service via a software factory to place within the network that accepts data from the non-conforming device, marshaling it to the external receiver. Figure 2b indicates this alteration to the network. After making this change, reanalysis using the script results in success.

Although we have currently not implemented an automated process for generating additional services in this way, it aligns closely to the Autonomous Computing paradigm [9]. Using such techniques allows the network to reorganise itself

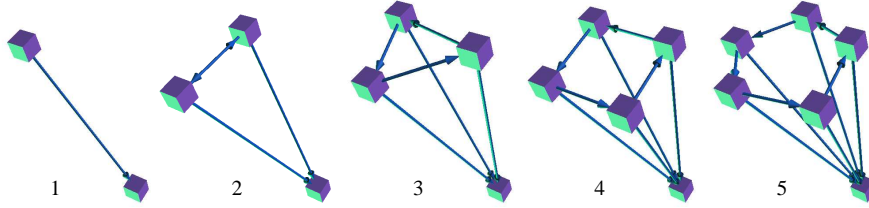


Fig. 3. Network topology for increasingly large networks.

dynamically in light of changes to the network structure, maintaining security in a reactive manner.

4 Complexity and Simulation Results

From the point of view of complexity the algorithm is dominated by the depth-first traversal of the nodes. This traversal algorithm has complexity $\mathcal{O}(|V| + |E|)$ where V is the number of vertices (nodes) of the graph and E is the number of edges (links). The checking algorithm can be seen to have complexity $\mathcal{O}(|E_v|)$ where E_v is the number of edges attached to vertex v . This checking algorithm must be undertaken for each external vertex. Since edges are directed (so will be counted only once for each vertex) and since the number of external vertices is bounded by V , the checking algorithm therefore has complexity bounded by

$$\prod_{v \in V_{\text{external}}} \mathcal{O}(|E_v|) \leq \prod_{v \in V} \mathcal{O}(|E_v|) \leq \mathcal{O} \left(\prod_{v \in V} |E_v| \right).$$

But $\prod_{v \in V} |E_v| = E$ and consequently the overall complexity of the algorithm is

$$\mathcal{O}(|V| + |E| + |E|) = \mathcal{O}(|V| + |E|).$$

We tested this and the effectiveness of the algorithm using the script provided above within the MATTS framework.

A series of networks were considered with an increasing number of nodes. In order to provide a consistent approach, internal nodes were arranged in a ring, with each internal node also attached to single external node as shown in Figure 3. The size of the network was increased by adding internal nodes to the ring, as can be seen. To ensure the script was completed in its entirety (thereby generating the maximum analysis time), the network was organised to satisfy the script, with all external links set as being encrypted. In the case when the network has a problem, the script would be likely to fail at an earlier stage, reducing the analysis time accordingly.

To provide accurate timings each test was performed 3000 times and average results taken. We ran the tests on a 663 MHz Intel Celeron machine with 256 Mb of RAM and running Windows XP Pro. We do not consider this to be ideal for

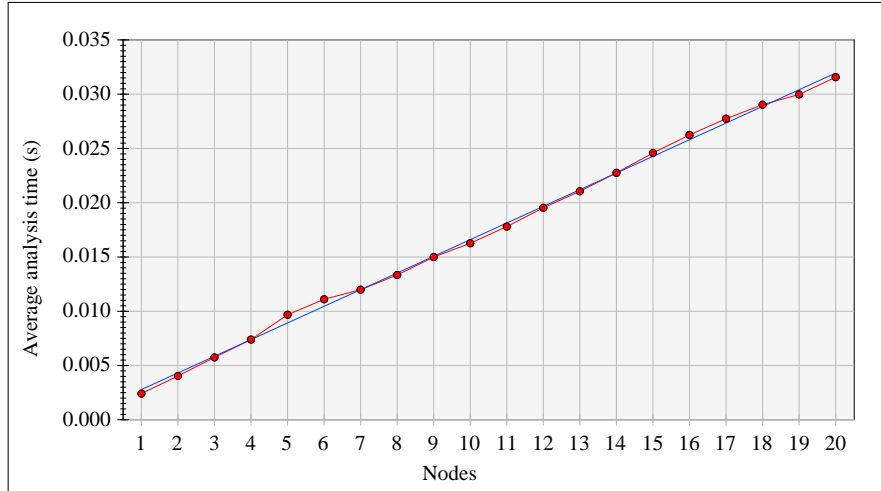


Fig. 4. Time taken to analyse boundary for increasingly large networks.

testing due to the unpredictability of the environment. However, whilst specific timing results may not be of particular relevance, the general trend established is of interest. The results can be seen in Figure 4.

There is a very clear linear correlation between nodes and analysis time, indicating that the analysis time is directly proportional to the number of nodes in the networks tested. The line of best fit – as indicated on the graph – is roughly

$$y = 0.0015494n + 0.001144.$$

where n is the number of nodes plotted along the x-axis. Looking at Figure 4 we can see that in these experiments $V = n + 1$ and $E = 2n$. Consequently the complexity in this scenario is expected to be $\mathcal{O}(n)$, fitting closely the experimental results observed.

5 Conclusion

We presented an algorithm for establishing the boundaries of a network that can be applied to dynamically changing environments. Such an algorithm has utility in enforcing security policies that rely on relationships between internal and external nodes, and on secure component composition properties such as Composable Assurance. We envisage that such an analysis might appropriately be used by the network infrastructure to determine policy satisfaction as a network changes dynamically.

Based on an analysis of the algorithm and reinforced by experimental results, we showed that application of the algorithm is unlikely to be overbearing in a reasonably sized network.

There is clearly further work to be undertaken in developing the technique into a genuinely useful process. Although we described a number of relevant application scenarios, we have not yet applied the technique to these scenarios in a realistic setting across a network. Our current work involves integrating the MATTS framework with a Networked Appliance framework in order to allow substantive testing of composition results to be undertaken in realistic settings. This would therefore admit such testing for the algorithm described here.

Development of appropriate means for establishing properties for individual components – relating to the internal/external distinction and properties such as the encryption property used above – is also required.

Our wider intention with this work has been to develop a bridge between current networks that rely on static boundaries to enforce security, and truly fluid and ubiquitous networks for which the perimeter model no longer applies. A longer term goal, therefore, is to apply new composition techniques to provide security evaluation for situations in which it no longer remains appropriate to consider boundaries at all.

References

1. Campbell, R., Al-Muhtadi, J., Naldurg, P., Sampemane, G., Mickunas, M.: Towards security and privacy for pervasive computing. In: *Software Security - Theories and Systems. ISSS 2002. Revised Papers*, 8-10 Nov. 2002, Tokyo, Japan, Springer-Verlag (2003) 1–15
2. Yocom, B., Birdsall, R.: Security gateways debut. *Business Communications Review* **33**(8) (2003) 20–9
3. Gordon, L.A., Loeb, M.P., Lucyshyn, W., Richardson, R.: Tenth annual csi/fbi computer crime and security survey. Technical report, Computer Security Institute (2005)
4. Sabater, J., Sierra, C.: Review on computational trust and reputation models. *Artificial Intelligence Review* **24**(1) (2005) 33–60
5. Haggerty, J., Shi, Q., Merabti, M.: Beyond the perimeter: the need for early detection of denial of service attacks. In: *Eighteenth Annual Computer Security Applications Conference*, Las Vegas, NV, USA, IEEE Comput. Soc (2002) 413–22
6. Qiu, Y., Zhou, J., Bao, F.: Mobile personal firewall. In: *IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications. Volume 4.*, Barcelona, Spain, IEEE Comput. Soc (2004) 2866–2870
7. Simmonds, P.: Users fight back by breaking the boundaries [corporate intranet security]. *Network Security* **2005**(6) (2005) 4–6
8. White, S., Hanson, J., Whalley, I., Chess, D., Kephart, J.: An architectural approach to autonomic computing. In: *Proceedings of the First International Conference on Autonomic Computing*, New York, NY, USA, IEEE Comput. Soc (2004) 2–9
9. Kephart, J., Chess, D.: The vision of autonomic computing. *Computer* **36**(1) (2003) 41–50
10. Shi, Q., Zhang, N.: An effective model for composition of secure systems. *Journal-of-Systems-and-Software* **43**(3) (1998) 233–44