The AAAI-17 Workshop on
Artificial Intelligence for Cyber Security
WS-17-04

Qualitative Reasoning about Cyber Intrusions

Paul Robertson,[1] Daniel Cerys,[1] Robert Laddaga,[1]
Robert Goldman,[2] Mark Burstein[2]

[1]DOLL Inc.
114 Waltham Street #14,
Lexington, MA 02421, USA

[2]SIFT
N. 1st Avenue, Suite 400
Minneapolis, MN 55401-1689

paulr@dollabs.com, dcerys@dollabs.com, rladdaga@dollabs.com,
rpgoldman@sift.net, burstein@SIFT.net

Abstract

In this paper we discuss work performed in an ambitious DARPA funded cyber security effort.The broad approach taken by the project was for the network to be self-aware and to self-adapt in order to dodge attacks. In critical systems, it is not always the best or practical thing, to shut down the network under attack. The paper describes the qualitative trust modeling and diagnosis system that maintains a model of trust for networked resources using a combination of two basic ideas: Conditional trust (based on conditional preference (CP-Nets) and the principle of maximum entropy (PME)).We describe Monte-Carlo simulations of using adaptive security based on our trust model.The results of the simulations show the trade-off, under ideal conditions, between additional resource provisioning and attack mitigation.

Introduction

The response to an attack depends upon what urgency exists for ongoing computation. The conservative approach would be to stop using the attacked system until it has been cleaned of malware, but this is not always wise, especially if the attacker wanted that outcome and the computation is essential for an important activity. Avoiding attack while handling attacks in such a way as to minimize loss and above all to continue is sometimes the preferred strategy.Our objective is to have systems deal with attacks in a most beneficial way by being aware of what has been attacked and self-adapting (Laddaga 1997, Laddaga et. al. 2001, Laddaga et. al. 2003 and Robertson et. al. 2000) to minimize the impact of the attack on the mission.This is an attempt to dodge the bullet.It accepts that occasionally an attack against the cyber infrastructure will succeed despite all attempts to protect against them and that when this happens the system must self-adapt in order to minimize the success of the attack. By "self–adapt", we mean the adjustment of the network infrastructure to ensure that the most trusted components of the network are used and that the most violated parts of the network are avoided.

This approach is a dynamic approach because once a network has been successfully attacked the attack can, in principle, spread at light speed to other cyber assets.The act of dodging the bullet is a continuous one.In this paper, we will not explore the larger problem of adaptation under attack but rather focus on an important part of the technology that addresses the state of trust of the cyber assets – what we call the trust model -- and the diagnosis of the cause.At any point in time we need to know what cyber assets are trusted more than others so that we can continuously adapt to be using the most trusted elements.Rather than attempt to measure the severity of an attack we endeavor to track the nature of the attack and its significance to mission tasks – does the attack diminish the reliability of the machine for the specific tasks it has been assigned.

Trust Modeling

The notion of trust is a familiar one. In many situations involving trust, a combination of machine learning and statistical methods can be useful.Our human experience shows that we model trust on experience. Experience with people, experience with situations, and experience with things.In a self adaptive cyber security environment most of these intuitions about trust are wrong.

When a configuration changes frequently by self-adaptation, any particular configuration is likely unique and the lifetime of the configuration is likely short making statistical approaches difficult to apply.Furthermore, since we are dealing with modeling the trust of computational components in the face of a malicious cyber attack, the trust has more to do with the attacker than with the computational component itself.There too lies a problem:The attacker must continually evolve in response to our continual attempts to thwart cyber attacks.Our evidence that we are under attack therefore is weak and our hypotheses about what the attacker will do next is similarly weak.

The essence of self-adaptive systems is that there are multiple ways of achieving a desired result and that we can change between these approaches in response to changes in the world.In the case of a networked computation system, we make have a choice between which hosts to use, for example, to run part of the computation. To support the needs of self-adaptation therefore, we need to be able to compare how much we trust one computational component (say a host) with another.For a computation that draws upon several communicating networked hosts (a configuration) we need to be able to compare one configuration against another candidate configuration in order to determine which configuration is more trustworthy.These kinds of relative qualitative trust support our ability to select an adaptation that will lead to a configuration that has higher trust in response to an ongoing cyber attack.Finally, there are occasions where relative qualitative trust alone is not sufficient and we need a measure of absolute trust.We sometimes need to ask when has our trust in a host, a program, or a configuration dropped to a point where immediate action is required.In the next two sections we give an overview of our implementation of qualitative trust and quantitative trust in the networked cyber domain.

We assume that we have explicit models of the computational assets that we are reasoning over including models of actual executing configurations and candidate alternative configurations and further that we are able to use these models to help identify unexpected behavior and diagnose which parts of the configuration may be responsible for the unexpected behavior.

In order to motivate the following discussions consider the following example of a networked computation – a mission whose goal is to accomplish video analysis for battle damage assessment.

After a bombing run, planes are sent to fly over the bombed targets taking video.The video is transmitted to a system that performs geo-location by matching against databases of terrain.The video along with the geo-location meta-data is streamed to another system that performs orthorectification.Finally the adjusted data and the accompanying meta-data are sent to an analyst's computer where the analyst can carefully examine each bombed target and assess the success of each hit. In this simplified example there are several computer systems configured in a network running specialized programs involving special databases that together perform a complex set of computations on the video.There may be several sites worldwide capable of performing these operations, some may be co-located, some may have different costs both monetary and performance.For example, using co-located systems may support faster communication between hosts but if that location has been hit by a cyber attack using systems spread around the world may offer greater trust but a higher communication cost.When all else is equal, we would prefer the configuration that has the least monetary cost and the shortest turnaround time but in the face of a cyber attack, as our trust in our ideal configuration declines we may be able to keep the computation going by selecting a more expensive configuration that offers greater trust.

Qualitative Trust

Our approach to qualitative trust builds heavily upon the ideas of relative desire (Doyle et al 1991), conditional preference, ceteris paribus (McGeachie et. al.), and CP-Nets (Boutilier et. al.).

By qualitative trust we refer to the idea that we can represent our level of understanding of trust in non quantitative ways that is nevertheless useful in making key decisions – such as choosing one host over another.

Conditional Trust

I trust Mr. X completely to keep a secret but I can't trust him at all to manage a budget.Mr. Y on the other hand loves to gossip and cannot be trusted with a secret but is quite reliable when it comes to adhering to a budget.When a machine is compromised it may be trusted less in general, but it may continue to be more trustable for certain kinds of operations.Our principle objective with this program is to keep a computation going despite determined cyber attacker's attempts to stop it.Instead of simply shutting down systems that may be distrusted we instead attempt to continue using the best that we can the systems that we have.It is therefore important to be able to manage not just a single trust value but rather to be able to represent trust in a particular capability.This idea of conditional trust draws directly on the idea of conditional preference (Boutilier et. al.).

Contagion and Trust Models

Since attacks are constantly evolving the knowledge of attacks that we can usefully apply are limited and very general.The notion of contagion is one such very general and useful approach to modeling trust.The idea is as follows.If host A running program B is diagnosed as having been attacked, certainly we trust it less that another host that has shown no signs of attack.Once a host has been attacked it may allow an attacker to attack a second host on the same network, or a host that the host is communicating with.Sometimes if a program P has been successfully attacked, it is the program itself that is less trusted and other sites on other networks may be trusted less because they are running the same program.This effect, which we call 'contagion', models the spread of distrust and suggests that the amount of distrust is greater in proportion to the number of hops to an affected host.Our system permits trust propagation rules to represent this kind of contagion, and leverage our model of a running configuration to better inform us of the state of trust of machines, programs, operating systems, and communication channels that have not even shown any form of deteriorated performance (yet).Note that during an ongoing attack we may easily reach a situation where contagion results in (almost) all assets have some level of distrust.Even when all assets suffer some degree of distrust, we can still function because our approach largely depends upon comparative trust and the cost of changing between configurations of a network computation.

D(attack(a,*))>D(attack(b,*)):: a >tr b (1)

Rule (1) above captures the unconditioned case of contagion.If D(…) is the shortest distance in 'hops' from an entity and a system that has been diagnosed as violated, this rule shows that a is trusted more than b if all else is equal if the distance from an attacked asset of a is greater than that of b.

Conditional Trust Models

Certain tasks are more sensitive to certain kinds of attacks than others.We can relate attack sensitivity by task with rules that indicate attack types that a specific task is less sensitive to:

AttackS. (Task= t1,{{at1|at2},{at3|at4},*}) (2)

AttackSensitivity rules relate attack types to tasks.Rule (2) above says that task t1 is less concerned about attacks of type at1 or at2 than it is of attack types at3 or at4 which in turn is less serious than all other types of attack represented by '*'.

Similarly, we can introduce rules that relate varied sensitivities to modes of contagion:

ContS (Task=t1,{{sprog,sos},{snet},{shost}}) (3)

Rule (3) says that for task t1 contagion by 'same program' or 'same operating system' is less worrisome than an successful attack on the same local network, which in turn is less worrisome than a successful a successful attack on the same host.That is to say if the same program running on a different network is successfully attacked, it is less problematic from the standpoint of running task t1 than, for example, an attack on another asset running on the same local network.

Whereas rule (1) gave a general contagion rule, we can specify more specific contagion rules by task, for example:

Task=t1&(4)

D(attack(a,{at1|at2}))>D(attack(b,{at1|at2})) ::a >tr b

Rule 4 implements the standard contagion rule explicitly for attack types at1 and at2 for task type t1.

More complex rules can be specified using our qualitative trust algebra.A network trust model is compiled from these rules along similar lines to that of CP-Nets [1].

Given the trust model and a diagnosis of attack diagnoses the self-adaptive program (not described in this paper) is able to ask questions of relative trust for given tasks as follows:

::hosta >tr hostb(General Query Asset)

Is hosta trusted (in general) more than hostb?

For a specific set of tasks:

Task={task1|task2}::hosta >tr hostb(task sp.)

Is hosta trusted more than hostb for performing task1 or task2?

In addition, whole configurations can be compared:

::configa >tr configb (configuration)

Is configuration configa trusted more than configb in general? While configuration details are not elaborated here, a configuration plan includes task information where appropriate, so that, as you would expect, the trust computation for a configuration takes into account the tasks that are to be performed by various assets in the configuration.

Additionally, a complex mission model will often include phases: data collection, data processing, and data dissemination, for example could be three phases of a mission model.Using rules like those provided above, we can additionally specify trust preferences on a per mission phase basis and then condition configuration trust preferences on the phase in question:

Phase={phase1| phase2}::configa >tr configb

If we are in either phase1 or phase2 do we trust configa more than configb?

By only specifying relative sensitivities to different forms of contagion and attack types conditionalized on task types and mission phases we are able to make relative trust comparisons that is sufficient to adapting a running configuration to a more trustworthy one when conditions change.

If a host is diagnosed as having been successfully attacked a chain of potential contagions will cause other assets to be trusted less than other depending upon the task and mission phase.We can simply compute which assets have their trust changed as the result of a malicious attack which allows the self-adaptive engine to ask the right questions in determining if an adaptation is possible that will minimize the impact on the mission of the observed attack.

Active Diagnosis

We have discussed what to do when we have confirmed an attack on a component; we propagate the reduced trust to other downstream components that are likely to candidates for attack.That is not enough, however.We would like to diagnose who the attacker is, because we can take actions to blacklist that attacker thereby minimize further damage being caused by the same attacker against other of our assets. If our component is on the edge and communication with external computers with whom we have no information, such as would be the case for a web services component, we will have to dig deeper to produce a list of the connections by interrogating log files, for example.Such sleuthing can be expensive, so we would not want to do it routinely, but given a confirmed attack, the effort is worthwhile.We assume, therefore, for every component type a collection a specialized taskable sensors that can be invoked to gather information.There are two different kinds of information that can be collected, the first concerns the clients of the attacked component and the second concerns an analysis of the traffic between each client and the attacked component that might cause the client to be suspected of being the attacker.

Diagnosis proceeds by generating a list of hypotheses about the attacker.If the attacked component is an edge component, specialized sensors are tasked to find the clients.One the full candidate list is established the evidence against each client is established.In general initially nothing will be known and the clients will all be equally likely.Specialized sensors for detecting suspiciousbehavior of these clients will be tasked to gather more information for the clients.At each stage the clients will be sorted in descending order of likelihood for causing the attack.On each iteration, the specialized sensors will adjust the trust in the clients and eventually one or more of the clients will rise in likelihood until a threshold is reached that incriminates the client enough for it to be blacklisted.The gathered evidence remains for the other clients so that if the attack continues after the blacklisting of a client, the diagnosis can continue to find other attackers.

For certain kinds of attack, such as a DDOS attack, there will be more than a single attacker.The attackers will thus be found one at a time and blacklisted until all clients have an acceptable level of trust.

This approach depends upon having specialized sensors that can be tasked by the diagnosis engine upon demand.These sensors can scan application logs in order to extract client identity and further look for activity profiles that do not match those of well behaved clients.

High Level Overview

In this section we present a worked example of the approach and how the trust model allows the self-adaptive system (not described in this paper) to perform adaptations in response to an evolving cyber attack.

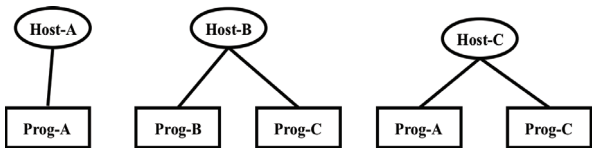

Figure 1 Cloud Resources

Figure 1 shows a simple cloud with three available hosts with different capabilities.Host-A can run program-A, host-B can run programs B and C while host-C can run programs A and C.

For a given mission we have three candidate configurations: shown in Figure 2.

Initially all assets are trusted.Based on non-trust related reasons, configuration A is preferred and established (Configuration A balances the load better than the other two configurations).At some point during the execution Host-A is successfully attacked.This results in the following ordering of configurations on the basis of configuration trust:

Config-C >tr Config-B >tr Config-A

The running system is therefore adapted to configuration C.This involves Running program-A on host-C (instead of host-A).Why is Configuration C better than configuration B?By contagion host-B is trusted less than host-C, configuration B uses host-B to run two programs whereas configuration C uses host-B only to run one program.

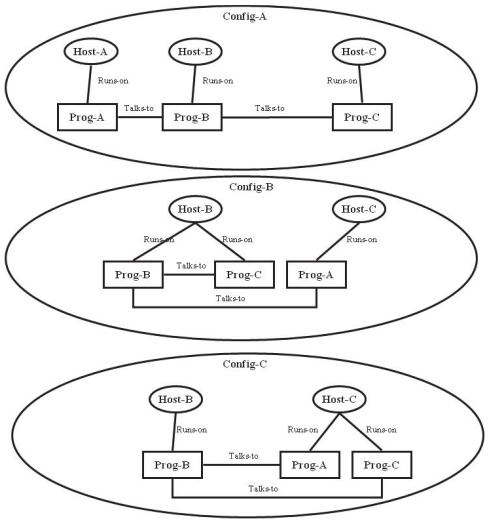

Figure 2: Potential Configurations

Results

In this paper, we have addressed a way of reasoning about relative state of trust of components of a distributed, networked, application.The eventual success depends upon the quality of the evidence provided by the sensors that must reason about unexpected behavior of the system. These aspects will be covered in forthcoming papers from the project.Here we address the best case that can be expected when the sensors are 100% accurate.

For this idealized example, we assume the ability to protect a host by providing a backup that can be switched to instantaneously, and the ability to fishbowl a host to deceive an attacker and thereby slow down an attack. What we wish to know is the minimum number of additional resources we need to protect the network fully or to bound the downtime due to successful attack.

In our simulation, we assume that a host once successfully attacked takes one time unit to recover.We additionally assume that a fishbowl provides one time unit of protection from attack.Finally we assume a mission that has 10 steps each taking one time unit.The simulation mounts a random attack during each time unit.A successful attack on each time unit will therefore have a worst case impact of doubling the mission time because a one time unit recovery is necessary during each time unit.A fully protected system will finish in 10 time units because all attacks could be dodged.

Run	Budget	Budget Percent	Mission Time
9	40	100	20
8	41	104	20
7	43	107	15.748
6	44	111	14.247
5	46	115	12.583
4	47	119	11.902
3	49	122	11.033
2	50	126	10.66
1	52	130	10.296
0	53	133	10.159

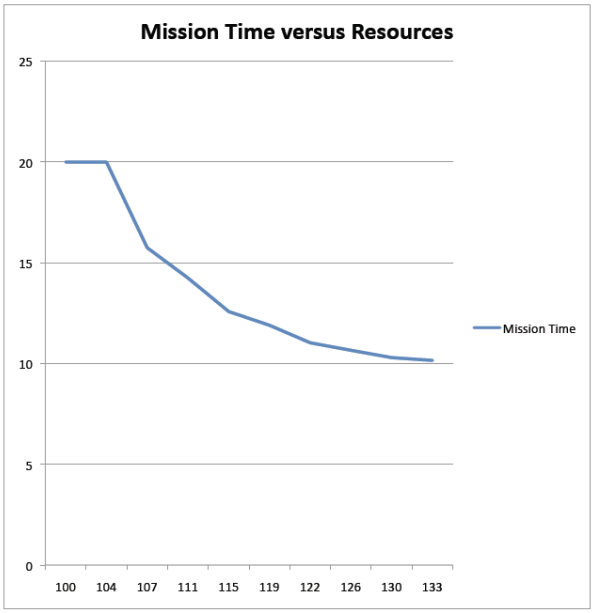

Figure 3: Simulation of the 10 step plan

Figure 3 shows the results of a Monte-Carlo simulation of the 10 step plan using randomly generated network configurations of 30 nodes with an average connectivity branching factor of 4. The results show that for a resource budget of 126% of the minimum resources, when provided with accurate sensor data, the system can be protected from most attacks.

The actual level of redundancy can be estimated for a given configuration.What is important here is to observe that a system can be protected from attack, if intelligent sensors providegood evidence, with a modest investment in additional resources.

Conceptually a system that has exactly the minimum resources required to get the job done is very brittle to any failure be it a natural failure or an deliberate attack.A modest over-provisioning of resources in a system that has dynamic reconfiguration and health sensors for the components, can survive in the face of deliberate and sustained attacks.

Conclusions

In this paper, we have focused on the use of qualitative reasoning about networked computations.The notion of relative trust supports the self-adaptation of networked computations so that the resources in use are always the most trusted resources for the job that is required of them.Given conditional trust, a resource that is no longer trusted to perform certain functions, such as be a SQL server, may be used to perform other non-SQL functions. Computations that have been performed on hosts that have become untrusted can be transferred to other resources that are more trusted for the operations in question.A compute resource that has thereby lost all of its computations in favor of other resources can be taken off line and reflashed to a fresh state at which point it can be reintroduced into the network and once again be a candidate for use in implementation parts of the mission.We have observed that a system constituted in this manner can remain running despite an ongoing and determined attack.We have shown that we can detect and diagnose zero day attacks and blacklist the attackers.

While avoiding an attack and keeping the attackers at bay is always the preferred route, there will always be cases where the determined attacker can get through our defenses.In such cases, the use of qualitative reasoning about trust in harmony with a diagnosis engine and specialized taskable expensive sensors, we can keep vital computations alive in spite of an onslaught of attackers.

Finally, we have spoken only of qualitative means in this paper but there are cases where quantitative information is also required.In fact our system includes both kinds of information, and the quantitative reasoning will be the subject of a follow-on paper.

Acknowedgements

This work was supported by Contract FA8650-11-C-7191 with the US Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory.The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Boutilier, C. Brafman, R. I., Domshlak, C., Hoos, H. H. and Poole, D. CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements

http://www.jair.org/media/1234/live-1234-2225-jair.pdf

McGeachie, J. & Doyle, J. Utility Functions for Ceteris Paribus Preferences
http://www.csc.ncsu.edu/faculty/doyle/publications/uc02.pdf

Jaynes, E. T. Probability Theory – The Logic of Science, Cambridge – (Chapter 12).

Doyle, J., Shoham, Y. M.and Wellman, P., 1991 A Logic of Relative Desire, Proceedings: Methodologies for Intelligent Systems, pp16-21.

De Kleer, J. and Williams, B. C. Diagnosing Multiple Faultshttp://www2.parc.com/spl/members/dekleer/Publications/Diagnosing%20Multiple%20Faults%20AIJ%20reprint.pdf

Morris, P. and Pearl, J., 1994 A Maximum Entropy Approach to Nonmonotonic ReasoningPAMI March 1994 Volume 15 #3 pp220-232

Laddaga, R., 1997 Self-adaptive software DARPA BAA 98-12. December.

Laddaga, R. Robertson, P and Shrobe, H. (Ed). 2001 Results of the First International Workshop on Self Adaptive Software.In Self-Adaptive Software.

Laddaga, R. Robertson, P and Shrobe, H. (Ed), 2003 Self-Adaptive Software: Applications, Volume 2614 Lecture Notes in Computer Science. Springer-Verlag.

Robertson, P., Laddaga, R. and Shrobe, H., 2000 Self-Adaptive Software, Volume 1936 Lecture Notes in Computer Science. Springer-Verlag.