ISSN:0975-9646

Using Network Security Management to solve Boolean Satisfiability Problem

Mekala Bhaskar, G.VishnuMurthy, V.Amarnath
Department of CSE, Anurag Group of Institutions,
CVSR College of Engineering, jodimetla, Ghatakesar, Hyderabad, AP, India

ABSTRACT- Enterprise network security management is a complex task of balancing security and usability, with trade-offs often necessary between the two. Past work has provided ways to identify intricate attack paths due to misconfiguration and vulnerabilities in an enterprise system, but little has been done to address how to correct the security problems within the context of various other requirements such as usability, ease of access, and cost of countermeasures. This paper presents an approach based on Boolean Satisfiability Solving (SAT Solving) that can reason about attacks, usability requirements, cost of actions, *etc.* in a unified, logical framework. Preliminary results show that the approach is both effective and efficient.

General Terms
Boolean Satisfiability Problem (SAT), Computer Network Management, Computer Network Security, Risk Analysis, Security, *Scalability*

1. INTRODUCTION

In computer science, **satisfiability** (often written in all capitals or abbreviated **SAT**) is the problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to TRUE. Equally important is to determine whether no such assignments exist, which would imply that the function expressed by the formula is identically FALSE for all possible variable assignments. In this latter case, we would say that the function is unsatisfiable; otherwise it is satisfiable. For example, the formula *a* AND *b* is satisfiable because one can find the values *a* = TRUE and *b* = TRUE, which make *a* AND *b* TRUE. To emphasize the binary nature of this problem, it is frequently referred to as *Boolean* or *propositional satisfiability*.SAT was the first known example of an NP-complete problem. That briefly means that there is no known algorithm that efficiently solves all instances of SAT, and it is generally believed (but not proven, see P versus NP problem) that no such algorithm can exist. Further, a wide range of other naturally occurring decision and optimization problems can be transformed into instances of SAT. A class of algorithms called SAT solvers can efficiently solve a large enough subset of SAT instances to be useful in various practical areas such as circuit design and automatic theorem proving, by solving SAT instances made by transforming problems that arise in those areas. Extending the capabilities of SAT solving algorithms is an ongoing area of progress. However, no current such methods can efficiently solve *all* SAT instances.

To make things more complicated, requirements for usability are often at odds with those for security. Configuration management would be a trivial problem if one only needed to consider security requirements; simply shutting down the whole network would resolve any security issues. But configuration changes aimed at correcting security flaws must be made in a context-aware manner, carefully balancing the system's security and usability. Existing works in enterprise network security analysis, such as MulVAL [19], [20], can identify all possible attack paths in an enterprise system and output them in a graph structure. This structure provides a good foundation for addressing how to automatically find the best way to correct the security problems presented in the analysis results. We have developed a systematic approach, shown in Figure1, to aid a human in confronting these difficulties. The current (problematic) network configuration settings are

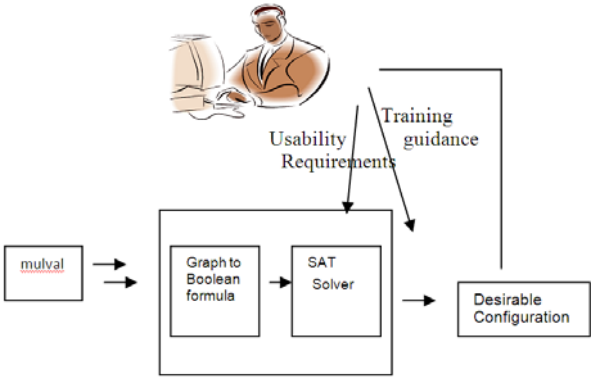

Fig. SAT-based configuration generation

passed into the MulVAL toolkit, which produces a logical proof graph identifying all potential attack paths by which an attacker might exploit system resources. This proof graph is converted into a Boolean formula in conjunctive normal form that relates configuration settings and attacker actions with potential effects, such as an attacker being able to execute arbitrary code on a computer in the network. Security and usability requirements, provided by the human user, are also converted into conjunctive normal form and added to the Boolean formula, and this combined formula φ is processed by a SAT solver.

A human user can further train the SAT solver as to the relative value of various system resources and usages.Working interactively, the human user is able to quickly identify and resolve network security issues without unknowingly lessening the system usability. As the tool is trained, the degree of automation should increase, producing sound and desirable reconfiguration suggestions with minimal human involvement. In this approach, we use two SAT solving techniques:

1) MinCostSAT can utilize user-provided discrete cost values, associated with changing a given configuration setting or allowing an attacker a given amount of access, to find a mitigation solution that minimizes the cost in terms of both security risk and usability impairment.
2) By examining the unSAT core, a minimal set of configurations and policy requirements that conflict, wenarrow the complexity of a reconfiguration dilemma to a straightforward choice between options. Past policy decisions by the human user are placed in a partialorderlattice and used to further reduce the scope of the decisions presented to the user.
By this approach, the human user is not expected to fully comprehend the effects, both good and bad, of all aspects of network configuration, but only to make decisions on the

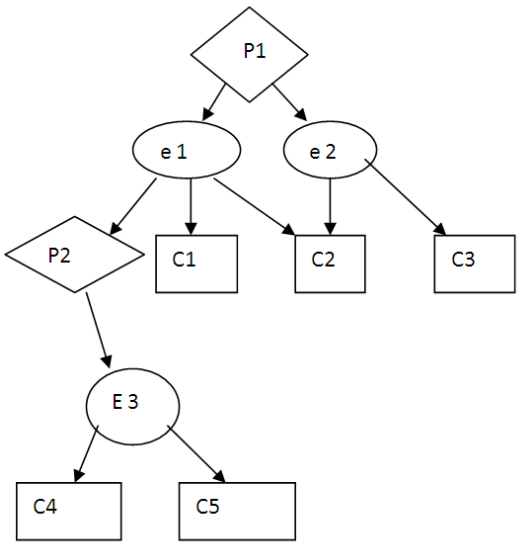

Fig. 2. A MulVAL proof graph

Immediate relative value of specific instances of usability and security. In this way, we reduce an extremely complex problem to one of more manageable proportions, automating the verification of both security and usability policies while introducing a method by which conflicts can be quickly and verifiably resolved.

II. MULVAL SECURITY ANALYZER

We use the MulVAL tool suite [19], [20] for our work.MulVAL is a security analyzation tool that, given initial network configurations (machines, active services, inter-host reach ability, *etc*.) and a database of known vulnerabilities, can identify all potential attack paths by which an attacker can exploit the system. These attack paths are assembled in a logical proof graph, showing how potentially successful attacks into the network are enabled by initial attacks on the outer edges. MulVAL's reasoning engine is specified declaratively in Datalog [1], providing inherent soundness of the results as well as an efficient $O(N2)$ running time [19].

Figure 2 shows part of the proof graph for an example enterprise network we studied. The diamond-shaped nodes in the graph represent privileges an attacker can gain through the exploits depicted as the elliptical nodes. System configuration data are represented by the rectangular nodes, such as *c*1, *c*2, *c*3, *c*4, *c*5. These can be both administrator defined configuration settings, like host access permissions, and unintentional facts, such as an existing vulnerability in a specific application. The potential exploits - *e*1, *e*2, *e*3 - link the causality relationship between a privilege that an attacker can gain and the preconditions that make this possible. For example, node *e*1 could correspond to a remote buffer overflow attack on a service. It links the effect of the attack, *p*1 (which means the attacker can gain privilege on the victim machine), to pre-conditions for the attack, such as *c*1 (which could mean the existence of a buffer-overflow vulnerability in
the service program), and *p*2 (which could mean the attacker's ability to send a maliciously crafted packet to the vulnerable service). All the arcs coming out of an exploit node like *e*1 form a logical AND relation, requiring all of its children to be true before this exploit can be used. The arcs coming out of a privilege node like *p*1 form a logical OR relation, in which multiple descendant nodes indicate alternative exploits by which an attacker can gain this privilege.
Although we have chosen to build our implementation based on the MulVAL proof graph, our approach can be based easily on other, similar tools for the production of network attack graphs (or fault propagation models) [6], [7], [11].

1.1.1 2-satisfiability

Main article: 2-satisfiability
SAT is also easier if the number of literals in a clause is limited to 2, in which case the problem is called 2SAT. This problem can also be solved in polynomial time, and in fact is complete for the class NL. Similarly, if we limit the number of literals per clause to 2 and change the AND operations to XOR operations, the result is *exclusive-or 2-satisfiability*, a problem complete for SL = L.
One of the most important restrictions of SAT is HORNSAT, where the formula is a conjunction of Horn clauses. This problem is solved by the polynomial-time Horn-satisfiability algorithm, and is in fact P-complete. It can be seen as P's version of the Boolean satisfiability problem.
Provided that the complexity classes P and NP are not equal, none of these restrictions are NP-complete, unlike SAT. The assumption that P and NP are not equal is currently not proven.

1.1.2 [Edit] 3-satisfiability

3-satisfiability is a special case of *k*-satisfiability (*k*-SAT) or simply satisfiability (SAT), when each clause contains exactly $k = 3$ literals. It was one of Karp's 21 NP-complete problems.
Here is an example, where ¬ indicates negation:

$$E = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4)$$

E has two clauses (denoted by parentheses), four variables (x_1, x_2, x_3, x_4), and k=3 (three literals per clause).
To solve this instance of the decision problem we must determine whether there is a truth value (TRUE or FALSE) we can assign to each of the variables (x_1 through x_4) such that the entire expression is TRUE. In this instance, there is such an

assignment (x_1 = TRUE, x_2 = TRUE, x_3=TRUE, x_4=TRUE), so the answer to this instance is YES. This is one of many possible assignments, with for instance, any set of assignments including x_1 = TRUE being sufficient. If there were no such assignment(s), the answer would be NO.

3-SAT is NP-complete and it is used as a starting point for proving that other problems are also NP-hard. This is done by polynomial-time reduction from 3-SAT to the other problem. An example of a problem where this method has been used is the Clique problem. 3-SAT can be further restricted to One-in-three 3SAT, where we ask if *exactly* one of the literals in each clause is true, rather than *at least* one. This restriction remains NP-complete.

1.1.3 There is a simple randomized algorithm due to Schöni 2-satisfiability

Main article: 2-satisfiability

SAT is also easier if the number of literals in a clause is limited to 2, in which case the problem is called 2SAT. This problem can also be solved in polynomial time, and in fact is complete for the class NL. Similarly, if we limit the number of literals per clause to 2 and change the AND operations to XOR operations, the result is *exclusive-or 2-satisfiability*, a problem complete for SL = L.

One of the most important restrictions of SAT is HORNSAT, where the formula is a conjunction of Horn clauses. This problem is solved by the polynomial-time Horn-satisfiability algorithm, and is in fact P-complete. It can be seen as P's version of the Boolean satisfiability problem.

Provided that the complexity classes P and NP are not equal, none of these restrictions are NP-complete, unlike SAT. The assumption that P and NP are not equal is currently not proven.

1.1.4 [edit] 3-satisfiability

3-satisfiability is a special case of *k*-satisfiability (*k*-SAT) or simply satisfiability (SAT), when each clause contains exactly $k = 3$ literals. It was one of Karp's 21 NP-complete problems.

Here is an example, where ¬ indicates negation:

$$E = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4)$$

E has two clauses (denoted by parentheses), four variables (x_1, x_2, x_3, x_4), and k=3 (three literals per clause).

To solve this instance of the decision problem we must determine whether there is a truth value (TRUE or FALSE) we can assign to each of the variables (x_1 through x_4) such that the entire expression is TRUE. In this instance, there is such an assignment (x_1 = TRUE, x_2 = TRUE, x_3=TRUE, x_4=TRUE), so the answer to this instance is YES. This is one of many possible assignments, with for instance, any set of assignments including x_1 = TRUE being sufficient. If there were no such assignment(s), the answer would be NO.

3-SAT is NP-complete and it is used as a starting point for proving that other problems are also NP-hard. This is done by polynomial-time reduction from 3-SAT to the other problem. An example of a problem where this method has been used is the Clique problem. 3-SAT can be further restricted to One-in-three 3SAT, where we ask if *exactly* one of the literals in each clause is true, rather than *at least* one. This restriction remains NP-complete.

There is a simple randomized algorithm due to Schöning (1999) that runs in time $(4/3)^n$ where n is the number of clauses and succeeds with high probability to correctly decide 3-Sat. The exponential time hypothesis is that no algorithm can solve 3-Sat in time $\exp(o(n))$.

ng (1999) that runs in time $(4/3)^n$ where n is the number of clauses and succeeds with high probability to correctly decide 3-Sat. The exponential time hypothesis is that no algorithm can solve 3-Sat in time $\exp(o(n))$.

III. RECONFIGURATION USING SAT SOLVING

Since any network misconfiguration is technically resolvable (if only by removing all inter-machine access), reconfiguration decisions must be made in consideration of the cost of the changes needed and of usability requirements. We have developed two approaches based on advanced SAT solving techniques that can automatically suggest optimal configuration changes to address the security problems presented in a proof graph. Our approaches allow a user to provide feedback to the SAT solver so that constraints on usability, cost of deployment, and potential damage due to successful attacks can all be optimized in a unified framework.

A. Transforming proof graphs to Boolean formulas

We first extract the causality relationships represented in a MulVAL proof graph and express them as a Boolean formula. This is best explained through an example. In the dependency proof graph of Figure 2, the AND node $e1$ means that the remote exploit is successful, since all of its children nodes $p2$, $c1$, $c2$ are enabled, and the result of the exploit is that the attacker gains privilege $p1$.

This can be expressed by the following formula,

$$p2 \Lambda c1 \Lambda c2 \Rightarrow p1$$

Or, equivalently,

$$\neg p2 \vee \neg c1 \vee \neg c2 \vee p1$$

We similarly convert the other exploit nodes to construct the Following formulae:

$e1 = \neg p2 \vee \neg c1 \vee \neg c2 \vee p1$

$e2 = \neg c2 \vee \neg c3 \vee \neg c5 \vee p1$

$e3 = \neg c4 \vee \neg c5 \vee p2$

Let $\varphi = e1 \Lambda e2 \Lambda e3$, then φ is a Boolean formula in conjunctive normal form (CNF) whose size is linear in the size of the proof graph1. φ encodes all the causality relationships between configuration data and potential attacker privileges shown in the proof graph. For example, if all of $c1$, $c2$, $c3$, $c4$, $c5$ are assigned the truth value T (as in the current configuration), then $p1$, $p2$ must be assigned T to make a satisfying assignment for φ. Therefore, if one wishes $p1$, $p2$ to be false (meaning an attacker can gain neither of these Privileges), at least some of $c1$, $c2$, $c3$, $c4$, $c5$ must be assigned F, meaning some of the current configuration settings need to be changed. Let $\psi = \varphi \Lambda \neg p1 \Lambda \neg p2$; then seeking a satisfying assignment to ψ amounts to finding configuration settings that can prevent an attacker from gaining privileges $p1$, $p2$.

Every variable representing a configuration setting will be assigned *T* (meaning that the setting is "enabled") or *F* ("disabled").
Since every configuration setting is *T* ("enabled") when the proof graph is constructed, removing or "disabling"
that setting will negate the associated variable. For example, if $c1$ represents the existence of a software vulnerability on the web server, the negation of that node means patching the vulnerability; if $c5$ represents a reach ability relationship between the Internet and the VPN server, disabling that node means blocking that access. If we feed ψ to a SAT solver, we can get a satisfying assignment by simply disabling all
the configuration nodes $c1$, $c2$, $c3$, $c4$, $c5$. This is certainly not an optimal solution; we need a secure configuration that maintains basic network usability. A careful observation of the proof graph shows that by disabling $c5$ without altering $c1$, $c2$, $c3$, $c4$, we can prevent all
the attack paths in the system, but we must consider the effects of this decision. It is not necessarily the case that a minimal number of system changes represent the optimal
Reconfiguration. Suppose again that $c5$ represents accessibility of the VPN server from the Internet. Removing this access would certainly block an attacker, but it would also prevent legitimate users from remotely logging into the network via the VPN server. This type of trade-off between security and usability is often present in system configuration management. In configuring an enterprise network, we want to compare not only the potential cost in damage from a successful attack, but also the potential losses arising from decreased network usability. If the cost of completely securing the network against attackers is much higher than the potential losses from attacks, it could be a better solution simply to acknowledge and tolerate the possibility that an attacker can obtain some minor privileges on the enterprise system. In this example, we may decide that an optimal solution would not force $p2$ to be false, so we can redefine our goal to be $\psi = \varphi \ \Lambda \neg p1$.We must now re-examine the proof graph in light of this new ψ. Suppose that $c1$ and $c3$ represent vulnerabilities present in system applications. By patching these two vulnerabilities, we can disable these two nodes and thus eliminate all attack paths that could enable an attacker to gain privilege $p1$.This configuration would negate $p1$ without violating φ, so it satisfies ψ.Though it is relatively easy to examine and reconfigure this small example, a reliable and automated approach is needed to address security concerns in real-size enterprise networks.
We now introduce two applications of SAT solving to resolve network misconfigurations by balancing costs and potential damage.

B. MinCostSAT

MinCostSAT is a SAT problem which minimizes the cost of the satisfying assignment [9]. Mathematically, given a Boolean formula ψ with n variables $x1, x2, \ldots, xn$, each with cost $ci \geq 0$, find a truth-value assignment $X \in \{0, 1\}n$ such that *S*atisfies ψ and minimizes

$$C = \sum_{I=1}^{n} cixi$$

Where $xi \in \{0, 1\}$ and $1 \leq i \leq n$.
MinCostSAT has been thoroughly studied by the SAT solving community [2], [5], [9], [14]. Although the problem is NP-hard, modern SAT solvers have been very successful in practice, being able to handle Boolean formulas with millions of variables and clauses in seconds. We use the MinCostChaff solver [5] which is a MinCostSAT solver based on the zChaff SAT solver [13].
The MinCostSAT problem minimizes the cost for variables that are assigned *T* . This matches the semantics for privilege variables, whose *T* assignment means an attacker can gain some privilege and thereby cause some damage. But for configuration variables, the cost would be incurred when it is disabled, or assigned *F*. To model this correctly, we first transform our formula to use the negation of a Boolean variable to represent each configuration node. This way, when the variable is assigned *T* , it means that the corresponding configuration node is disabled, which will incur some cost.
With the expressiveness of Boolean formulas and the power of a SAT solver, a system administrator can ask questions like "what is the best way to reconfigure my system if I want to guarantee that the file server will not be compromised?" This can be done by forcing the Boolean variable x that corresponds to the privilege ExecCode (fileServer, someUser) to be false (i.e., conjoining $\neg x$ to the original formula). He can also ask questions like "Can I make the file server secure while allowing the web server to be accessed from the Internet?" We have implemented mechanisms that allow a system administrator to specify those additional constraints for the various queries he would like to conduct. Those constraints can be straightforwardly specified in Datalog and automatically transformed into additional clauses in the Boolean formula to be solved by the MinCostSAT solver. This kind of constraint can also become a part of the configuration policy. For example, a user might decide that the web server must be accessible from the Internet. If the variable representing this
Configuration setting is forced true in the Boolean formula; MinCostSAT will never return a suggested reconfiguration that requires this access to be removed. Similarly, potential attacker privileges can be forced to be always false; for example, a user might decide that an attacker should never access the data historian, and so this access could be forced to be false, meaning that MinCostSAT will never allow it to be true. This effect could also be simulated by assigning unrealistically high costs for those variables; however, forcing them to be true or false will ensure that no reconfiguration suggestion will reverse this decision.

C. Scalability

To test the scalability of our approach, we constructed simulated enterprise networks with two different sizes
I: 100 host machines, evenly divided in 10 subnet
II: 250 host machines, evenly divided in 25 subnets
We also tested using two different cost functions:
A: All clauses were assigned an equal cost. The effect of this cost policy would simply be to minimize the number of configuration changes made plus the number of compromised machines.

B:Clauses represent in code-execution privileges on a machine were assigned costs based on the machine's position in the network. The effect of this cost policy would be to have increasingly high costs for penetrations deeper into the network. The costs for blocking network access
to hosts or disabling network services were significant. All other changes had equal, low cost.
The test was conducted on a Linux machine with Opteron Dual-Core 2214 2.2 GHz CPU, with 16GB memory, and running Gentoo Linux with kernel version 2.6.18-hardenedr6

Sz	Cfn	#variables	#clauses	time(sec)
I	A	11,853	12,053	0.11
I	B	11,853	12,053	0.21
II	A	70,803	72,553	3.03
II	B	70,803	72,553	6.49

The simulated networks on which we performed the above tests were certainly not representative of realistic enterprise network settings, but the performance indicates that modern SAT solvers are likely to be powerful enough to handle the configuration management problem we describe in this paper. Also, a highly correlated network configuration may produce nontrivial runtimes. A full-scope understanding of the scalability of this approach will require extensive real-world testing, currently planned for future work.

D. Iterative UNSAT Core Elimination
We now introduce the second SAT solving technique, in which the concept of UNSAT core is leveraged for the identification and resolution of conflicts in the network policies.
Definition 1. An unsatisfiable core is a subset of the original CNF clauses that is unsatisfiable in itself [4].When a SAT solver finds a set of clauses to be unsatisfiable,a byproduct of this decision is the UNSAT core. Logically, given an unsatisfiable Boolean formula ψ in CNF, the UNSAT core $\mu = u1, u2, \ldots, um$ is a subset of all the clauses in ψ (shorthanded μ □ ψ hereafter) such that ψ will remain unsatisfiable while μ remains unchanged. We generate the UNSAT core using the zChaff SAT solver's zcore function [13].In this approach we will not rely on cost assignments, but rather on the balance between security and usability policies

Returning to the example from section II, let security policy $\delta = -p1$; then our security policy specifies that an attacker should not be able to gain privilege $p1$. Let usability policies $\gamma 1 = c1 \wedge c2 \wedge c4 \wedge c5$ and $\gamma 2 = c2 \wedge c3 \wedge c5$; then our usability policies together specify that all current configuration settings are necessary to maintain basic network usefulness. So $\psi = \varphi \wedge \delta \wedge \gamma 1 \wedge \gamma 2$.

Utilizing the UNSAT core in this way precludes the need to assign costs to each network configuration setting beforehand, as is required for the MinCostSAT solution. So long as security and usability policies do not conflict, the user is not asked to decide between any two policies or attempt to assign discrete values to them. These decisions are only faced when an actual conflict has arisen, so the human user makes only necessary choices about system resource valuations.

Partial-order lattice: To further reduce the breadth of decisions faced by a human user, we have implemented a partial-order lattice to store the relative priorities between pairs of policies. Each time the human user is presented with the causes of an unclassifiablegy conflict and selects one or more of those constraints to be relaxed, this decision is recorded in the partial-order lattice to be used as a reference for deciding future conflicts. We assume that the constraints. That the user allows being relaxed has a lower overall priority than any clauses that were not relaxed, and this ordering is recorded in the lattice. In future decisions where two conflicting constraints appear for which an ordering is already known, the constraint with higher priority will not be offered to the user as a possibility for relaxation. In this way, conflicts
are reduced to comparisons between configuration settings or policy requirements for which relative priorities are not known. Once known, these decisions need not be faced again

CONCLUSION

We have introduced a methodology where the system security requirements can be converted to a Boolean formula and, using SAT solving techniques, one can quickly correct
Misconfigurations that may lead to multi-step, multi-host attacks in enterprise networks. This approach can account for both security and usability requirements, through the adoption of modern SAT solving techniques such as MinCostSAT and UNSAT core elimination. We presented a unified framework in which the competing requirements can be specified in a
Boolean formula and an optimal solution can be searched for that provides a reasonable trade-off between the various requirements for practical security administration. Preliminary
Experimental results on both realistic and synthesized enterprise network settings indicate that the SAT solving approach is effective and scalable.
Definition 2.
Let $x_1, \ldots, x_m$ be Boolean variables. We define an *literal* to be either x_i or $\neg x_i$, for $1 \leq i \leq m$. We define a *clause* to be the joining of some number of literals, by a $\vee$, the "logical or", surrounded by parenthesis. That is, a clause is $(c_1 \vee \cdots \vee c_k)$, with c_i a literal, for $1 \leq i \leq k$. Finally, let us define a *formula* as joining some number of clauses with $\wedge$, the "logical and". That is, a formula is $c_1 \wedge \cdots \wedge c_n$, where c_i is a clause, for $1 \leq i \leq n$.
We can now define the **boolean satisfiability problem**: Given a formula S with boolean variables $X = \{x_1, \ldots, x_m\}$, decide if there exists a function $f : X \to \{true, false\}$, such that when one replaces x_i by $f(x_i)$ in S, the resulting boolean sentence is logically true. We note this as $f(S) = true$, and say that S is *satisfiable*.
This may seem like it applies only to a very restricted subset of Boolean propositions; however, any Boolean proposition can be reduced to one of these formulas. This is the conjunctive normal form of a Boolean proposition. Thus, for any Boolean proposition S you have, there exist

REFERENCES

Davis, M.; Putnam, H. (1960). "A Computing Procedure for Quantification Theory". *Journal of the ACM* **7** (3): 201. oi:10.1145/321033.321034.

Davis, M.; Loge Mann, G.; Loveland, D. (1962). "A machine program for theorem-proving". *Communications of the ACM* **5** (7): 394–397. doi:10.1145/368273.368557.

Cook, S. A. (1971). "The complexity of theorem-proving procedures". *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*: 151–158. doi:10.1145/800157.805047.

Michael R. Garey and David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman. ISBN 0-7167-1045-5. A9.1: LO1 – LO7, pp. 259 – 260.

Marques-Silva, J. P.; Sakallah, K. A. (1999). "GRASP: a search algorithm for propositional satisfiability". *IEEE Transactions on Computers* **48** (5): 506. 10.1109/12.76943

[1] Stefano Ceri, George Gottlob, and Letizia Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Trans. Knowledge Data Eng.*, 1(1):146–166, 1989.

[2] Olivier Coudert. On solving covering problems. In *33rd Design Automation Conference (DAC'96)*, pages 197–202, 1996.

[3] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[4] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Proc. Theory Applications Satisfiability Testing - SAT 2006*, pages 252–265, 2006.

[5] Zhaohui Fu and Sharad Malik. Solving the Minimum-Cost Satisfiabilityproblem using SAT based branch and bound search. In *Proc.International Conference on Computer-Aided Design (ICCAD)*,San Jose, CA, USA, 2006.

[6] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida, December 2006.

[7] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches andChallanges*, chapter 5. Kluwer Academic Publisher, 2003.

[8] Somesh Jha, Oleg Sheyner, and Jeannette M.Wing. Two formal analyses of attack graphs. In *Proc.15th IEEE Computer Security Foundations Workshop*, pages 49–63, Nova Scotia, Canada, June 2002.

[9] Xiao Yu Li. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. PhD thesis, North Carolina State University, Raleigh, North Carolina, 2004.

[10] Richard Lippmann, Kyle Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Mike Artz, and Robert Cunningham. Validating and restoring defense in depth using attack graphs. In *Military Communications Conference (MILCOM)*, Washington, DC, U.S.A., October 2006.

[11] Richard Lippmann and Kyle W. Ingols. An annotated review of past papers on attack graphs. Technical report, MIT Lincoln Laboratory, March 2005.

[12] Richard P. Lippmann, Kyle W. Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Michael Arts, and Robert Cunningham. Evaluating and strengthening enterprise network security using attack graphs. Technical Report ESC-TR-2005-064, MIT Lincoln Laboratory, October2005.

[13] Yogesh S. Mahajan, Zhaohui Fu, and Sharad Malik. Zchaff2004: An efficient SAT solver. In *Lecture Notes in Computer Science SAT 2004Special Volume*, pages 360–375. LNCS 3542, 2004.

[14] Vasco M. Manquinho and Jo ao P. Marques-Silva. Search pruning techniques in SAT-based branch-and-bound algorithms for the binatecovering problem. *IEEE Trans. Computer-Aided Design*, 21:505–516,2002.

[15] Vaibhav Mehta, Constantinos Bartzis, Haifeng Zhu, Edmund Clarke, and Jeannette Wing. Ranking attack graphs. In *Proc. Recent Advances in Intrusion Detection (RAID)*, September 2006.

[16] Sanjai Narain. Network configuration management via model finding.In *Proc. 19th conference on Large Installation System Administration Conference (LISA)*, 2005.

[17] Sanjai Narain, Gary Levin, Vikram Kaul, and Sharad Malik. Declarative infrastructure configuration synthesis and debugging. *J. Network Systems and Management, Special Issue on Security Configuration*, 2008.

[18] Steven Noel, Sushi Jajodia, Brian O'Berry, and Michael Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs.In *19th Annual Computer Security Applications Conference (ACSAC)*, December 2003.

[19] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference onComputer and Communications Security (CCS)*, pages 336–345, 2006.

[20] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. Mul-VAL: A logic-based network security analyzer. In *14th USENIX Security Symposium*, 2005.

[21] Reginald Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *13th European Symposium on Research in Computer Security (ESORICS)*, October 2008.

[22] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29:3812–3824, November 2006.

[23] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using attack graphs. In *Third Workshop on Quality of Protection(QoP)*, 2007.