Investigating Survivability of Configuration Management Tools in Unreliable and Hostile Networks

Karvinen, T. and Li, Shuliang

Karvinen, Tero and Li, Shuliang. Investigating Survivability of Configuration Management Tools in Unreliable and Hostile Networks. Proceedings of the 3rd International Conference on Information Management (ICIM 2017), Chengdu, China. 21st - 23rd April 2017. IEEE.

Investigating Survivability of Configuration Management Tools in Unreliable and Hostile Networks

TERO KARVINEN
Digital Business
Haaga-Helia University of Applied Sciences,
Ratapihantie 13, FI-00500 Helsinki,
FINLAND
tero.karvinen@iki.fi
University of Westminster,
35 Marylebone Road, London NW1 5LS,
UNITED KINGDOM

SHULIANG LI
Westminster Business School,
University of Westminster,
35 Marylebone Road, London NW1 5LS,
UNITED KINGDOM
lish@westminster.ac.uk
School of Economics & Management,
Southwest Jiaotong University,
Chengdu, Sichuan 610031, CHINA

***Abstract* — A configuration management system (CMS) can control large networks of computers. A modern CMS is idempotent and describes infrastructure as code, so that it uses a description of the desired state of a system to automatically correct any deviations from a defined goal. As this requires both complete control of the slave systems and unquestioned ability to provide new instructions to slaves, the private key of the master is highly valuable target for attackers. Criminal malware networks already survive in hostile, heterogeneous networks, and therefore, the concepts from those systems could be applied to benign enterprise CMSs. We describe one such concept, the hidden master architecture, and compare its survivability to existing systems using attack trees.**

Keywords - configuration management system; survivability; attack tree; command and control; botnet

I. INTRODUCTION

Configuration management systems (CMSs) have proliferated to meet the challenges of growing sizes of computer networks with more hosts per administrator, heterogeneous networks, cloud [1] and grid computing, stricter requirements for verified security and faster response time to markets using DevOps methods [2]. A CMS is an essential part of large computer installations, as they can extend life, reduce cost, reduce risk, and even correct defects [3].

Modern configuration tools are versionable and idempotent. In practice, the configuration manifests describing the target state of the network must be plain text to be stored in a version control system. This “Infrastructure as Code” approach allows administrators to use software engineering methodology to control their network [4]. Idempotence means that an operation can be applied multiple times without changing the result beyond the initial application.

CMSs are based on the master-slave architecture. The master computer will issue configurations, and the slave computers apply these configurations. Thus, a successful attack on a master results in full compromise of every slave it controls. This makes it a very tempting target for attack and therefore, protecting CMSs gets a high priority.

II. SURVIVABILITY

Survivability is the capability of a system to fulfill its mission in the face of challenges. These challenges or faults include attacks by human adversaries, system failures, accidents and failures of large parts of network infrastructure [5, 6, 7]. These challenges or faults are often meant to cover all potential damaging events to the system [6].

The purpose of the system is essential when evaluating survivability. Even if individual components stay functional, the system has failed if it fails to provide the intended service it was designed for. Identification of essential services is the key concept of survivability. Essential systems are those that either are required for meeting the mission requirements or those whose failures threaten the system. [8] A CMS meets the criteria of an essential system, as a compromise of the CMS results in full compromise of all controlled systems. Also, without a CMS it becomes difficult or even impossible to react to changing environments and threats in a timely manner.

Information security is traditionally defined as the CIA triad: confidentiality, integrity and availability. Confidentiality means that confidential information is not exposed to unauthorized parties. Integrity is the assurance that data is not modified without permission. Availability means that systems respond to users in timely manner [9]. Some writers consider information security a subcategory of survivability [10].

Survivability analysis is the process of identifying components susceptible to attacks, then quantifying the capability to survive these attacks [5]. One method of identifying suitable targets for attack is the attack tree method. To compare attack trees, we’ll look at both malware and CMS network architectures.

Karvinen, Tero and Li, Shuliang. Investigating Survivability of Configuration Management Tools in Unreliable and Hostile Networks. Proceedings of the 3rd International Conference on Information Management (ICIM 2017), Chengdu, China. 21st - 23rd April 2017. IEEE.

III. Malware Command and Control Network Architectures

Botnet is a network of compromised machines under the control of an attacker [11]. Largest botnets have had millions of slave computers in hostile, heterogeneous environments. As these botnets are extensively documented, they can provide insights into possibilities of novel network architectures for CMSs. Even though the CMS tools could be used for malware CC and vice versa, in this paper we categorize tools by their intended or most common purpose.

Malicious botnets have both similarities and differences to benign configuration management tools. Both aim to provide a scalable, resilient and timely updates to slave configuration. But only botnets require secrecy in each point: slave, network and possible CC server. Botnets might also require the ability to reduce their forensic footprint, function in more heterogeneous environments and withstand legally sanctioned attacks against their CC infrastructure and unpredictably changing network conditions. Botnets are sometimes used for extracting data from victim systems, and this puts additional strain on secrecy requirements. Once the CC analysis has been published, interested parties can create new intrusion detection system (IDS) rules and antivirus detection routines [11].

Command and control channels have seen improvements over time. Some earlier botnets, such as Agabot, SDBot and SpyBot, used Internet relay chat (IRC) as a control channel. Network operators can attempt to automatically detect botnet activity, making uncommon protocols hard to hide. This has made HTTP a tempting choice for bots such as BlackEnergy, Rustock and Clickbot. [11]

Gu considers IRC a push architecture, because commands are immediately sent to slaves as the botmaster sends them [12]. It should be noted that in the client-server architecture, the slaves (bots) are still clients when they connect to the IRC server. In this way, they can access the master trough NAT and firewall.

At its peak, Zeus botnets had infected 3.6 million computers in just the US. As Zeus was a banking malware, it caused significant damage. In Zeus botnet, slaves use direct HTTP connection to pull catalogs from CC server. Because Zeus is crime-ware, a tool to build these botnets, multiple parties have created their own botnets and their CC infrastructures. Thus, destroying a single set of CC servers is not enough to disable Zeus. [11]

For a truly distributed operation, some bots have adopted peer-to-peer (P2P) architecture. In P2P, slaves connect to each other without a central server. In TCP/IP sense, each node can act as either server or client, as dictated by network conditions. Dittrich (2008) names Peacomm and Nugache as examples of P2P botnets [13]. In addition to the lack of server as a single point of failure, he mentions small network footprint and unpredictable traffic patterns as additional benefits.

Conficker.C was a highly advanced botnet in 2009. Variants of Conficker infected millions of machines and saw constant updates and move to more stealthy operation. Conficker.C used a time based algorithm for locating peers, and thus avoided the need for initial seed list of peers. [14]

Stuxnet was a botnet to attack Iranian nuclear facilities. It is the first successful cyber attack with physical damage of this scale [15]. The target factories enjoyed military protection, the computers were air gapped and the virus operated in a country that likely was highly suspecting of the makers of the malware. The version of Stuxnet caught for analysis used a direct HTTP connection to pre-programmed servers [16].

Naz was the first bot using social networks as a CC channel. It uses steganography to hide control messages on Twitter, pretending to be a human user. Other social networks and third party services could be used as a CC channel by new botnets. [17]

Cryptolocker and similar software encrypt user files and extort for money. The ransom must be paid in Bitcoin to receive the key to decrypt the files. Modern encryption extortion can even work without a traditional CC network, as the password is passed for human to type after criminals have received payment.

This look on the CC networks of successful malware indicates some trends in protocols and architectures. Considering protocols, many networks use standard protocols, especially HTTP/HTTPS. It could be speculated that this is driven by existing tooling and skill in these common protocols, ability bypass even the most restrictive firewalls and the possibility to hide in the noise of existing traffic using the same protocols. The architectures seem to remove single points of failure and distance the actual master (with signing keys to control slaves) from direct connection to slaves, while still securing the catalogs against hostile modification.

To recognize malware CC concepts not yet applied to CMSs, we look into the state of CMS network architectures.

IV. Implied Direct Master-Slave Connection

To send the instructions over a network such as Internet, typical systems use client-server architecture between a master and slaves. The server can reside on the slave (push) or on the master (pull). Many articles on CMS assume that slaves must at some point directly contact the master. This assumption can be either explicit or, more often, implicit. As we have seen in the examination of successful malware CC architectures, such direct contact is not an absolute requirement.

In their comparison of open source CMSs, Delaet considers only two possible network architectures: push and pull. He explicitly states on page 6: “In all approaches, each managed device contains a deployment agent that can be push or pull based” [18].

Karvinen, Tero and Li, Shuliang. Investigating Survivability of Configuration Management Tools in Unreliable and Hostile Networks. Proceedings of the 3rd International Conference on Information Management (ICIM 2017), Chengdu, China. 21st - 23rd April 2017. IEEE.

Vanbrabant [19] categorizes "deployment architectures" of CMSs to pull or push, implying direct network connection between a master and slaves. This is further emphasized by his examples. Poat et al. [20] have selected popular CMSs (Chef, Puppet, CFEngine) for comparison, all of which require direct connection between a master and slaves to configure multiple computers. In their paper on orchestration ("model-driven Cloud management") Wettinger et al. [1] imply direct connection between a master and a server in their choice of tools and in the options they use for the orchestration system to deploy the catalogs. Even though there are multiple peer-reviewed works on applying the configuration, less interest is paid on the secure transport of these configuration instructions. For example, Swięcicki[21] describes a novel tool "Overlord", but bypasses the transfer by stating that the "program then could be transported to the target machine".

Practical CMSs in the industry are using direct push and pull architectures, too. We briefly compared modern, free software CMSs. For the purposes of this work, tools using idempotent configuration with infrastructure as code were considered modern. Tools whose licensing met both the criteria of the Open Source Initiative (OSI) and Free Software Foundation (FSF) were considered free software.

We collected a list of candidates from literature survey [22], articles referenced in this paper (e.g. [20]) and non-academic sources. The list was then filtered to exclude dead projects, those failing to meet the criteria of free, idempotent and infrastructure-as-code. To identify key tools, the interest to those tools was evaluated by estimating both fresh (since 2015) academic references using Google Scholar and non-academic search traffic based upon Google Trends data. Both approaches gave similar results. The key CMSs were Puppet, Chef, Ansible and Salt. If we would have considered older academic references for tools that still have large production installations, CFEngine would have been included in this list, too.

The key CMS Puppet, Chef, Ansible, Salt (and CFengine) all use either direct push or direct pull approach. They allow for local application of configuration, if the source code for configuration is securely transported to slaves. All key CMS can be configured to use any of pull, push or locally applied architectures, but they usually prefer and recommend one architecture over others. Puppet, Chef, Salt and CFengine recommend pull, Ansible recommends push.

Most key CMSs use common protocols for transfer. Puppet and Chef use HTTPS, specifically hypertext transfer protocol (HTTP) protected by transport layer security (TLS) with self-signed certificates. Ansible uses secure shell (SSH), namely OpenSSH. Salt uses more unique approach of ZeroMQ based protocol to allow very fast (non-idempotent) parallel command execution in addition to idempotent catalogs.

This overview of the academic articles and key CMSs has indicated that both literature and the key CMS tools assume direct pull or push connection between slaves and a trusted master server.

V. Hidden Master Architecture

As we have noted, combining catalog signing keys (root access to slaves) with catalog distribution causes security risks and other problems. Hidden master architecture avoids this problem by keeping the signing keys in a computer that only connects to the Internet to upload catalogs to an intermediate server.

These intermediate catalog distribution servers do not need to be secure. In fact, they can be commodity web servers in networks not controlled by an organization using a CMS. As distributing static files from a web server is very efficient, working as an intermediate distribution server for catalogs could be a side job for any computer with a web server.

Physical control of computers is a prerequisite for security: "boot access is root access". For example, an infrastructure or platform as a service (IaaS or PaaS) hosting provider usually has full access to guest systems. In traditional push or pull based systems, this precludes storing catalogs in inexpensive cloud providers such as Amazon, DigitalOcean or Linode.

In the hidden master architecture, having multiple low value, untrusted servers to distribute catalogs can remove catalog distribution as a single point of failure. Using inexpensive third party cloud providers improves survivability against problems that affect whole networks or geographical locations.

A. The Operation of the Hidden Master Architecture

Asymmetric encryption is used for securing the communication between the hidden master and the slaves. Each slave has a secret key to open the catalogs encrypted by the slave public key on the hidden master. These keys can be generated either on master or the slave as dictated by practical requirements when provisioning the systems.

The most valuable key is the hidden master private key, which is generated and always stays in the master computer. This key used for signing the instructions for each slave. All slaves have a copy of the related, trusted public key. The slaves blindly trust any instructions signed by the key of the hidden master. This key is protected by the hidden master architecture, as none of the slaves know how to reach the hidden master. The hidden master can stay offline during normal operation.

To command the slaves, operator uses the hidden master to compile catalogs of instructions to slaves, which are then encrypted using each slave's public key and signed with the hidden master's private key. These encrypted catalogs are transferred to untrusted intermediate hosts known to slaves. Each slave downloads these instructions when it's periodically checking the intermediate distribution hosts. If the slave can decrypt the catalog using its secret key and verify the signature using the hidden master's

public key, the instructions are then applied. Otherwise, the encrypted catalog is discarded.

We have performed initial experiments of implementing hidden master architecture with an existing key CMS. This reduces the amount of novel code, making it possible to improve survivability without developing a whole configuration management system with a resource abstraction layer and domain specific language.

B. Comparing Hidden Master Architecture to Push/Pull Attack Tree

Attack tree is a method of systematically categorizing all methods by which the system can be attacked [23, 24].

Attack tree takes the view of an attacker, and starts with the goal of compromising the target system. This is the root node of the tree. This node is then divided into subnodes by splitting the problem area. Each subnode is further split until all attacks have been enumerated. Each branch can consist of parts of attack that must all be achieved (AND) or alternative avenues of attacks (OR). [25, pp. 4–6]

Compromising the master server is the ultimate goal of the attack. Once the signing keys are obtained, the attacker fully controls all slaves. No further attacks would be needed, and the attacker can move on to fulfill his mission, such as exfiltration of data, launching further attacks on other networks, encrypting essential data for extortion or installing advanced persistent threats. Disabling a CMS will also make it difficult to react to threats and faults in a timely matter.

Figure 1 shows attack tree against regular pull architecture. Attack surface reduced by the proposed hidden master architecture is crossed out. The main challenge to survivability in regular pull architecture is the high value of the master server. On one hand, it needs heavy protection against attacks, thus limiting it to secure premises and computers without other software limiting attack surface. After all, a successful attack on the master server would mean compromising all slaves, making it the most valuable computer in the network it controls. On the other hand, in order to withstand disruptions in the network, master servers should be duplicated in multiple networks and geographical locations.

Even if some of these problems could be mitigated by creating completely separate configurations and master servers, this would soon mean losing single source of truth, making administration more expensive and error prone.

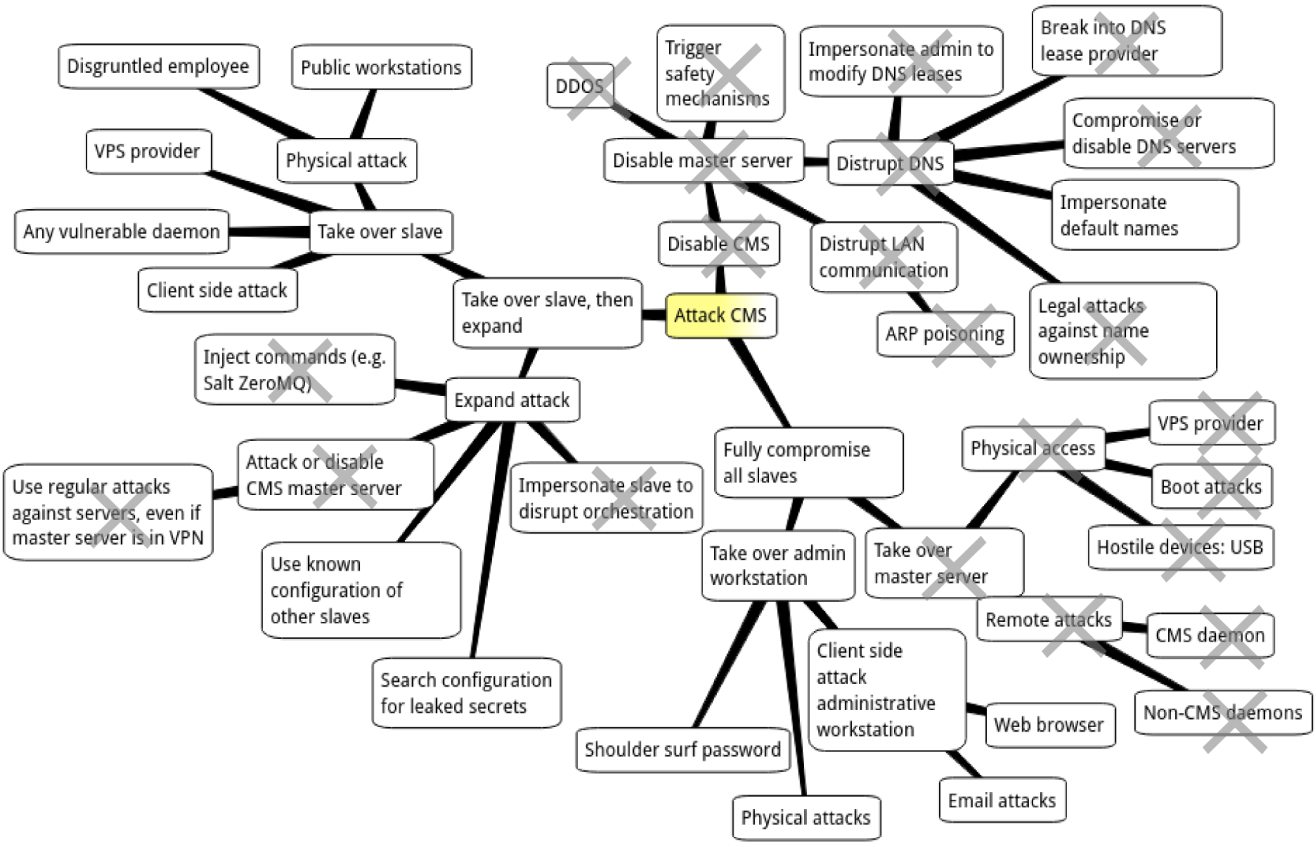

Fig 1: An attack tree with reduced attack surface crossed out

Karvinen, Tero and Li, Shuliang. Investigating Survivability of Configuration Management Tools in Unreliable and Hostile Networks. Proceedings of the 3rd International Conference on Information Management (ICIM 2017), Chengdu, China. 21st - 23rd April 2017. IEEE.

In the hidden master architecture, the servers contacted by slaves are not trusted. There can be large number of these servers, and the encrypted catalogs can be served as a side function of any low value web servers. Because the encrypted catalogs are simple files, distributing them around the globe and in different networks is very cheap. In fact, all slaves do not need to know all the places where new instructions can be provided, leaving some locations to work as backup for the most valuable slaves.

Receiving log data from slaves is left for future research. To use a similar methodology as in this paper, it could be looked what techniques malware uses for data exfiltration.

VI. Conclusions

Malware command and control networks survive in hostile and unreliable environment. Many CC networks use encryption over HTTP protocol and distance the actual master, that is, the owner of the trusted signing keys, from catalog distribution, so that slaves do not have access to master.

Current industry practice in CMSs is for slaves to directly contact single or few master servers directly. As the trusted signing keys reside on the master, they are clearly the most valuable computers in the network they control. The value of the masters requires very high level of both physical and software security, which poses challenge to survivability, as attempts to improve availability by duplicating the servers in different networks reduce the owner's control of these systems.

In this paper, we have proposed hidden master architecture to alleviate this problem. In hidden master architecture, the master with the trusted signing keys only contacts the network when uploading encrypted, signed catalogs to untrusted distribution servers. These untrusted distribution servers can be any commodity web servers. The slaves download the catalogs from any of these servers. If some intermediate servers are down or compromised, the slaves simply contact the next servers. Initial experiments have been conducted and the findings are encouraging. It is possible for us to enhance survivability without the need for developing a whole configuration management system.

References

[1] J. Wettinger, M. Behrendt, T. Binz, U. Breitenbücher, G. Breiter, F. Leymann, S. Moser, I. Schwertle, T. Spatzier, and others, "Integrating Configuration Management with Model-driven Cloud Management based on TOSCA," In: *CLOSER*, 2013, pp. 437–446.

[2] M. Rajkumar, A. K. Pole, V. S. Adige, and P. Mahanta, "DevOps culture and its impact on cloud delivery and software development," in *2016 International Conference on Advances in Computing, Communication, Automation (ICACCA) (Spring)*, 2016, pp. 1–6.

[3] N. Perera, "Automatic Configuration Management - Autodiscovery of Configuration Items and Automatic Configuration Verification," SpaceOps Conferences, 16-20 May 2016, pp.1-13.

[4] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" 2016, pp. 189–200.

[5] C. Fung, Y.-L. Chen, X. Wang, J. Lee, R. Tarquini, M. Anderson, and R. Linger, "Survivability analysis of distributed systems using attack tree methodology," in *MILCOM 2005 - 2005 IEEE Military Communications Conference*, 2005, Vol. 1, pp. 583–589.

[6] N. R. Mead, R. J. Ellison, R. C. Linger, T. Longstaff, and J. McHugh, "Survivable network analysis method," DTIC Document, 2000.

[7] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245–1265, Jun. 2010.

[8] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. A. Longstaff, and N. R. Mead, "Survivability: protecting your critical systems," *IEEE Internet Computing*, vol. 3, no. 6, pp. 55–63, Nov. 1999.

[9] M. Atighetchi and J. Loyall, "CrossTalk - Meaningful and Flexible Survivability Assessments: Approach and Practice," *The Journal of Defense Software Engineering*, pp.12-18. March/April, 2010.

[10] P. G. Neumann, "Practical Architectures for Survivable Systems and Networks,(Phase-Two Final Report)," *Computer Science Laboratory, SRI International*, 2000.

[11] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, "On the analysis of the Zeus botnet crimeware toolkit," in *2010 Eighth Annual International Conference on Privacy Security and Trust (PST)*, 2010, pp. 31–38.

[12] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," Proceedings of the 15th Annual Network and Distributed System Security Symposium, 2008.

[13] D. Dittrich and S. Dietrich, "P2P as botnet command and control: a deeper insight," in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, 2008, p. 41–48.

[14] N. Fitzgibbon and M. Wood, "Conficker.C : A Techical Analysis," Sophos Labs, Sophos Inc, Apr. 2009.

[15] R. Langner, "To Kill a Centrifuge - A Technical Analysis of What Stuxnet's Creators Tried to Achieve," The Langner Group, 2013.

[16] N. Falliere, L. O. Murchu, and E. Chien, "W32.stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, 2011.

[17] E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu, "Social network-based botnet command-and-control: emerging threats and countermeasures," in *Applied Cryptography and Network Security*, 2010, pp. 511–528.

[18] T. Delaet, W. Joosen, and B. Van Brabant, "A Survey of System Configuration Tools," In: *LISA*, 2010, vol. 10, pp. 1–8.

[19] B. Vanbrabant, "A Framework for Integrated Configuration Management of Distributed Systems (Een raamwerk voor geïntegreerd configuratiebeheer van gedistribueerde systemen)," 2013 IFIP/IEEE International Symposium on Integrated Network Management, 2013.

[20] M. D. Poat, J. Lauret, and W. Betts, "Configuration Management and Infrastructure Monitoring Using CFEngine and Icinga for Real-time Heterogeneous Data Taking Environment," *Journal of Physics: Conference Series* 664 (2015) 052020, 2015, vol. 664, pp. 1-6.

[21] B. Święcicki, "A Novel Approach to Automating Operating System Configuration Management," in *Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology – ISAT 2015 – Part II*, A. Grzech, L. Borzemski, J. Świątek, and Z. Wilimowska, Eds. Springer International Publishing, 2016, pp. 131–142.

[22] J. Hintsch, C. Görling, and K. Turowski, "A Review of the Literature on Configuration Management Tools," CONF-IRM 2016 Proceedings, 2016.

[23] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[24] S. Mauw and M. Oostdijk, "Foundations of attack trees," in *International Conference on Information Security and Cryptology*, 2005, pp. 186–198.

[25] A. P. Moore, R. J. Ellison, and R. C. Linger, "Attack modeling for information security and survivability," DTIC Document, 2001.