European Journal for Security Research (2021) 6:211–227
https://doi.org/10.1007/s41125-022-00079-7

ORIGINAL PAPER

Twin Based Continuous Patching To Minimize Cyber Risk

Fabrizio Baiardi[1] · **Federico Tonelli**[2]

Received: 15 June 2021 / Accepted: 3 January 2022 / Published online: 25 January 2022

Abstract

Digital twins are virtual replicas to simulate the behavior of physical devices before they are built and to support their maintenance. We extend this technology to cybersecurity and integrate it with adversary emulation to define a remediation policy that selects and schedules patches for the vulnerabilities of an information and communication infrastructure before threat actors can exploit them. Distinct twins model, respectively, the infrastructure and threat actors. The former twin describes the infrastructure modules, their vulnerabilities, and the elementary attacks actors can implement. The attributes of the twin of a threat actor describe its attack surface, its goals, how it selects attacks, and it handles attack failures. The Haruspex software platform builds the twins of the infrastructure and those of the threat actors, and it automates the emulation of an actor. In this way, it can discover the attack paths the actor implements without disturbing the infrastructure. In each path, the actor composes elementary attacks to reach its goal. Multiple emulations can discover all the paths of an actor by covering stochastic factors such as attack success or failure. The knowledge of these paths enables the remediation policy to minimize the patches to deploy. Since new vulnerabilities continuously become public, new countermeasures are needed. A twin-based approach supports a continuous remediation process to handle changes in the infrastructure, new vulnerabilities, and new threat actors because the platform can update the twins and run adversary emulations. If new attack paths exist, the platform applies the remediation policy. Experimental data confirm the effectiveness of this approach.

Keywords Model-based · Adversary emulation · Digital twin · Vulnerability · Patch schedule

✉ Fabrizio Baiardi
f.baiardi@unipi.it

Federico Tonelli
federico.tonelli@haruspex.it

[1] Dipartimento Di Informatica, Università Di Pisa, Pisa, Italy

[2] Haruspex Srl, Pisa, Italy

1 Introduction

A digital twin is a digital replica of a physical object or system. The technology first arose at NASA: a full-scale replica of space capsules, to mirror and diagnose on the ground problems in orbit. Later this resulted in fully digital twin simulations, and it has expanded to describe large systems, such as buildings, factories, and smart cities to evaluate properties of interest before building the system. In 2017 Gartner named digital twins as one of the top 10 strategic technology trends for that year, and it predicted that in at most five years "billions of objects will be represented by digital twins, a dynamic software model of a physical thing or system." In 2018 Gartner estimated "21 billion connected sensors and endpoints by 2020, digital twins will exist for billions of things in the near future." A digital platform takes an object twin together with data that describe a phenomenon involving the object and it uses these inputs to predict or simulate how the phenomenon will affect the object. Several fields currently adopt digital twins to support predictive maintenance and to keep assets in peak operating conditions and avoid unexpected breakdowns.

This paper describes how to apply the twin technology to manage the risk due to the discovery of new vulnerabilities in the off-the-shelf software and firmware modules of an information and communication technology, ICT, infrastructure, and it defines a remediation policy to select and schedule countermeasures to stop the threat actors, or at least to reduce their success probability, anytime new vulnerabilities become public. A long time to deploy countermeasures strongly increases risk because most attacks occur only after their enabling vulnerability becomes public. To minimize the deployment overhead, the remediation policy should minimize the number of selected countermeasures. The problem is critical due to the large number of vulnerabilities that become public. Even if a twin-based approach can support a wide class of countermeasures, for the sake of simplicity, this paper focuses on patches, the countermeasures that remove vulnerabilities by deploying new code to replace flawed one. Patches are among the cheapest countermeasures, but no effective solution exists to select the patches to deploy and to schedule their deployment. Therefore, the average time to patch a vulnerability is steadily increasing and it currently ranges from 60 to 150 days but with a very long tail. Further confirmation of the lack of effective policies is the compulsory direction the Cybersecurity & Infrastructure Security Agency, a federal US agency, has issued in November 2021. The direction lists the vulnerabilities federal agencies should patch and some of these vulnerabilities have been public for more than three years.

The proposed policy is risk-based. Risk-based policies schedule patches according to the risk each vulnerability poses, and they define metrics to evaluate this risk (Dey et al. 2015; Manzuik et al. 2006; Roytman and Jacobs 2019). Our remediation policy evaluates risk through attack paths (Baiardi and Sgandurra 2013; Baiardi 2015; Sarraute et al. 2011) i.e., by sequences of attacks that escalate the agent privileges till it controls the modules it is interested in. The risk depends upon the paths where a vulnerability appears, and the policy computes

the smallest set of patches to stop all the paths. We discover paths through twin-based adversary emulation (Applebaum et al. 2016, 2017; Eckhart and Ekelhart 2019; Moskal et al. 2018; Strom et al. 2018) and automate it through the Haruspex platform (Baiardi and Sgandurra 2013; Baiardi et al. 2013; Baiardi 2015). The platform uses the twins to run multiple independent adversary emulations to cover stochastic factors such as the success or the failure of an attack. To minimize the emulation overhead, the twin of the infrastructure is not a fully detailed replica of the infrastructure. Instead, it consists of an advanced asset inventory with information on the infrastructure modules, their interconnections, their vulnerabilities, and the corresponding attacks. An actor twin models a threat actor, and its attributes describe actor properties.

The platform can run on the target system or a distinct one, i.e., on a cloud (Theoharidou et al. 2013), and it supports continuous remediation i.e., continuous patching deployment, driven by vulnerability discovery. As soon as some vulnerabilities become public, the platform includes them in the infrastructure twin, and it runs the emulations in parallel with, and without disturbing, the normal working of the infrastructure. Anytime the new vulnerabilities open new paths, the platform applies the remediation policy. Hence, the proposed policy can be applied to industrial control systems and, more in general, to infrastructures with operational technology components.

This paper is organized as follows. Section 2 introduces some terminology and discusses current solutions to select and schedule patches. Then, Sect. 3 classifies the various agents and describes how the remediation policy computes a patch schedule based upon attack paths. Section 4 describes digital twins and how they support the discovery of attack paths through multiple independent adversary emulations. Section 5 introduces continuous remediation and the conditions that fire the execution of new adversary emulations. Lastly, Sect. 6 presents some experimental results of the proposed policy, and Sect. 7 draws some conclusions and discusses future developments.

2 Computing a Patch Schedule: Current Solutions

We briefly review state-of-the-art policies to compute a patch schedule. First, we introduce some terminology and then review some popular policies. In the following, infrastructure and actor denote, respectively, the target ICT infrastructure and a threat actor.

2.1 Terminology

A vulnerability is a weakness, error, defect, flaw, or bug that enables some attacks targeting the confidentiality, integrity, and availability of information (CVE https://cve.mitre.org; National Vulnerability Database, https://nvd.nist.gov/).

The infrastructure is the target of some threat actors. Information about each actor is known, and it includes its attack surface and goals. An attack surface includes all

the attacks the legal privileges of the actor enable. Each goal is a set of privileges on some modules the actor aims to acquire to control some critical assets. Threat actors seek advantage of vulnerabilities in hardware, software, and firmware, to attack the infrastructure. The greater the window of time between the discovery of a vulnerability and the remediation, the patching in our case, the more time an actor has available to implement an attack. Each attack is paired with a precondition and a postcondition that include, respectively, the privileges to execute the attack and those a successful execution of the attack grants. Due to the infrastructure complexity, a threat actor can reach a goal only by composing attacks in an attack path (Baiardi et al. 2013; Sarraute et al. 2011) that is in a sequence that escalates its privileges to those in one goal. After the success of all the attacks in an initial subsequence of a path, the actor owns the privileges to execute the next one in the path. A path is minimal if none of its proper subsequences is a path. A path is not minimal anytime the actor selects and executes a useless attack due to poor information on the infrastructure. A useless attack does not grant the privileges the actor needs to reach a goal. In the following, we say a vulnerability appears in a path if it enables an attack in the path.

2.2 Discovering Vulnerabilities

Any policy to select and deploy patches needs, at least, two databases. The first one is an inventory that lists all the hardware and software modules of the infrastructure. The second database lists the vulnerabilities of each module. The simplest way to build the two databases is by deploying in the infrastructure sensors that run a vulnerability scanning (Baiardi et al. 2013). The first step of the scanning runs a fingerprinting to discover the modules each node runs. An active sensor implements a fingerprint by sending malformed packets to some ports of a node. Then, it analyzes the node replies to discover the modules the node runs. Instead, a passive sensor sniffs and analyzes messages among nodes to recognize the modules that exchange these messages. Passive fingerprinting does not disturb the infrastructure, but it takes longer because it needs multiple messages to identify a module. However, it can collect these messages by continuously monitoring the infrastructure. Furthermore, high-performance sensors are not required because missing some messages is less critical than when monitoring traffic to detect intrusions. In general, passive sensors should be preferred because they do not add noise or computational load to the infrastructure. Since active sensors are more accurate, a compromise must be chosen between the noise due to the sensors and the accuracy of the information they return.

After identifying the modules, the second step of the scanning accesses some vulnerability databases (CVE https://cve.mitre.org; Martin 2019; National Vulnerability Database, https://nvd.nist.gov/) to discover their vulnerabilities. This may result in both false positives and false negatives because some vulnerabilities may have been patched or a module may be affected by some vulnerabilities that do not appear in the database. To minimize the number of false positives, some sensors execute at least one of the attacks a vulnerability enables. The attack success confirms the existence of the

vulnerability (Chuvakin and Barros 2018). This may damage the infrastructure or degrade its performances because the sensor behaves like a threat actor and the technology requires human assistance to minimize its impacts.

2.3 Computing a Patch Schedule

The knowledge of the infrastructure vulnerabilities suffices to compute a patch schedule only when adopting a *patch-all* solution. This popular policy suggests patching each new vulnerability as soon as a patch exists. The policy is often ineffective because it applies so many patches that most are yet to be deployed when new vulnerabilities become public.

Risk-based, or scoring, solutions rank vulnerabilities by mapping some of their attributes into a score that approximates the resulting risk. Then, they only patch the vulnerabilities with a score larger than a fixed threshold, starting from those with the largest scores. The most popular risk-based solution is the Common Vulnerability Scoring System, CVSS (Mell et al. 2006, 2007; Tripathi and Singh 2011) that maps the main characteristics of a vulnerability into a numerical score in the range from 0 to 10, the largest severity. In principle, CVSS consists of three metric groups: Base, Temporal, and Environmental. The first group represents the intrinsic qualities of a vulnerability that are constant over time and across user environments such as the complexity of the attacks the vulnerability enables and the privileges to implement them. The Temporal group reflects characteristics that change over time such as the availability of an exploit. It evaluates the likelihood that a vulnerability exists and the one an actor will exploit it. Lastly, metrics in Environmental group consider the infrastructure characteristics but they are seldom applied because their evaluation cannot be automated. Hence, the evaluation of the risk due to a vulnerability usually applies metrics in the Base group only and the CVSS defines a calculator to map attributes into a score.

Scoring solutions like CVSS focus on a vulnerability in isolation rather than on the whole infrastructure and they neglect how an attacker exploits the vulnerability in paths to a goal. Even a low score vulnerability results in a large risk if it is the missing link to chain two attack sequences into one path. On the other hand, a high score vulnerability does not change the risk if actors cannot acquire the privileges to launch any attack it enables. Experimental data on attacks confirm the importance of context-dependent information to discover how actors exploit vulnerabilities even when they have available accurate information on the infrastructure vulnerabilities. Hence, any scoring system independent of the target infrastructure only offers a rough, approximated value for a risk-based ranking. Lastly, any solution that considers each vulnerability in isolation can remediate a vulnerability only if a patch exists.

3 An Attack Path Scheduling Policy

This section discusses how the remediation policy uses information on attack paths to select and schedule patches. First, we classify actors and then discuss how to minimize the patches to deploy. Lastly, we propose a solution to increase the resilience

of the schedule. We detail in the next section how to discover the attack paths of an actor. For the moment being, we assume these paths are known and focus on the remediation policy.

3.1 Classifying Actors

We partition threat actors into adaptive and fixed.

An adaptive actor has explicit goals i.e., some modules it aims to control. The actor is rational and tries to minimize its effort to a goal. Furthermore, it reacts to both updates of the infrastructure and attack failures by implementing distinct attacks. Hence, the actor can manage an attack failure by executing a distinct path.

A fixed actor aims to control the largest number of infrastructure nodes to download a payload on each one. The payload is a software module that may steal information, implement a ransomware attack or connect the node to a botnet i.e., an overlay network the actor controls. To reach its goal, the actor chooses a target module and executes a set of attacks in a fixed order until one of them succeeds and the actor can replicate onto the node that runs the module. Then, the actor selects another target and repeats the procedure. The actor selects a new target even after a predefined number of failures of its attacks. At any time, multiple instances of the same actor can run on distinct nodes.

We pair each actor with a set of paths. For an adaptive actor, these are the paths it implements to its goal. For fixed actors, we assume there are some infrastructure nodes to protect and consider the paths to any of these nodes from the actor attack surface. The paths paired with an actor depend upon both vulnerabilities and the actor and they change anytime the infrastructure or the actor changes. As discussed in the following, this should also consider how adaptive actors react to the patching.

3.2 Stopping Threat Actors

The proposed policy neutralizes an actor by stopping all its paths. This requires patching at least one vulnerability that appears in each path. Before selecting the vulnerabilities to patch, the policy removes useless attacks from a path because the actor does not need them to reach the goal. It also removes a vulnerability if a corresponding patch is not available. A *reduced path* is the subset of the useful attacks in a path such that there is a patch for the enabling vulnerability. An optimal choice of the patches to deploy to stop a set of reduced paths Sp privileges shared vulnerabilities among the paths in Sp because patching each of these vulnerabilities stops multiple paths.

To minimize the number of patches, the policy computes a max–min patch set of Sp i.e., the smallest set of patches that targets at least one vulnerability in each path in Sp. The selection of a max–min patch set for Sp is an NP-hard problem (Mell et al. 2016). To schedule the deployment, vulnerabilities in the max–min patch set can be ranked according to the sum of the weights of the paths where each appears. The weight of a path depends upon the impact paired with the corresponding goal.

To stop multiple actors, Sp should include the reduced paths of all these actors. Even here the policy minimizes the patching overhead through shared vulnerabilities.

Besides minimizing patches to deploy, the main advantage of a policy focusing on paths rather than on a single vulnerability is that it can stop a path even if some vulnerabilities appearing in the path cannot be patched.

3.3 Building a Resilient Schedule

A policy that patches one vulnerability for each path may be ineffective if the actor knows some vulnerability that the vulnerability scanning does not return. Vulnerabilities that are unknown to the affected vendor are commonly referred to as zero-day vulnerabilities and they are viewed as the greatest prize for cybercriminals because they pose the greatest threat to information security. These vulnerabilities could enable an attack that replaces one stopped by the patches. A policy can increase the probability of stopping an actor that knows some zero-day by patching distinct vulnerabilities in each path. As an example, *k-patch* policies patch k vulnerability for each path (Wang et al. 2014). Even these policies exploit shared vulnerabilities.

4 Twin based Scheduling Solutions

We introduce the Haruspex platform (Baiardi and Sgandurra 2013; Baiardi 2015; Baiardi et al. 2013) and discuss how it builds the twin of the architecture and those of the actors. The platform automates adversary emulation and it runs multiple emulations to discover all the paths of the actors.

4.1 The Infrastructure Digital Twin

The choice of running multiple emulations implies that the infrastructure twin cannot be so detailed and describe the infrastructure modules at the instruction level. Hence, we have chosen a more abstract description that guarantees that *the twin has sufficient fidelity to allow the implementation of the desired security measure* (Eckhart and Ekelhart 2019). The infrastructure twin describes all and only those features of the infrastructure to emulate adversaries realistically and it consists of two databases that describe, respectively, assets i.e., modules, and vulnerabilities. The first database also includes information on the physical topologies, the logical one, filtering rules in the firewall, and routing rules in the gateways. Lastly, this database includes information on the trust relationship and the information flow among modules. A trust relationship implies that an actor that controls a module also controls those that trust this module. As an example, an actor that controls a hypervisor also controls the virtual machines running on the hypervisor. A typical information flow is the one between a web server and the database server with information the server accesses. An actor that controls the source of an information flow, the webserver in

the example, can transmit malicious information to the flow destination, the database server in the example.

The Haruspex platform collects the information to build the infrastructure twin by deploying active and passive sensors in the target infrastructure. The platform uses information from these sensors to populate the module database. Then, it builds the vulnerabilities database by accessing the Common Vulnerability Exposure database and the National Vulnerabilities Database (CVE https://cve.mitre.org; National Vulnerability Database, https://nvd.nist.gov/). The platform matches the vulnerability descriptions in these databases against some predefined patterns to extract attack attributes such as pre and post-conditions and the success probability. The emulation determines the success or the failures of an attack according to a probability distribution that depends upon the attributes of the attack and those of the actor twin.

4.2 Automated Adversary Emulation and Actor Twins

The Haruspex platform adversary emulation mimics the action of an actor to understand how they affect the infrastructure and improve its robustness. Hence, the abstraction level of the actor twin and its attributes should enable the emulation to reproduce any of these actions.

Attributes of an actor twin describe the actor attack surface, the resources it can access, its goals if any, and the information on the infrastructure the actor knows before starting its attacks. A critical attribute for the emulation is the strategy to select the action of an actor at a given time. For a fixed actor this strategy is very simple because it sequentially executes attacks in a predefined order independently of the target vulnerabilities. Hence, the emulation of these actors is not complex and their twin simply describes the attack sequence and the initial nodes that execute the actor.

The emulation of an adaptive actor is much more complex because the platform mimics the strategies it applies to select and execute an action and it determines the result, success or failure, of an action. Possible actions for an adaptive actor include, at least:

(a) collect information on the target infrastructure,
(b) select one of the possible attacks,
(c) select a target for the attack,
(d) implement the selected attack and handle its success or its failure.

When selecting an attack and a target, the infrastructure vulnerabilities and the current actor privileges constrain the possible choices of the strategy. Some strategies privilege attacks with a large success probability, while others prefer those returning the largest number of privileges. The strategy to handle an attack failure may repeat the attack or select another one. Distinct actors apply alternative strategies which, in turn, result in distinct sequences of actions. As an example, an insider already knows some information on the target infrastructure and so it can privilege

the selection and execution of an attack. Instead, an external attacker needs at first to collect some information to choose the best attack. Hence, this actor first discovers the IP addresses of some nodes and scans them for vulnerabilities. Some actors privilege the collection of information to acquire enough knowledge on the infrastructure to maximize the probability of selecting useful attacks only. Instead, other actors immediately attack a component as soon as they discover some enabling vulnerabilities at the expense of executing a useless attack.

The emulation of an adaptive actor can be described in terms of transitions of a finite state automaton where the state corresponds to the actor security state, the privileges, and the information on the infrastructure resulting from the previous actions of the actor. State transitions depend upon the strategies of the actor. The platform uses a distinct automaton for each actor twin.

A platform to automate adversary emulation should offer several alternatives for the mentioned strategies. As an example, the Haruspex platform can emulate, among others, actors that select attacks randomly and those that privilege, respectively, the attack with the largest success probability and the one returning the largest number of privileges. Another strategy ranks attacks according to their cost-performance ratio that depends upon the number of access privileges the attack grants and its success probability. Both deterministic and probabilistic strategies are supported and where the probabilities of selecting an action/a target may depend upon the security state.

The final output of adversary emulation in a (simulated) time interval is either an attack path, if the twin has reached one of its goals, or a failure. This assumes the platform updates the simulated time according to the time it takes to execute each action or to implement an attack. This time is a further output of the vulnerability analysis previously discussed.

Since the platform simulates both the success or the failure of an attack and how the actor handles a failure, the output of a single emulation is stochastic as it depends on both attack failures and how the actor handles this failure. Furthermore, distinct emulations of the same actor may discover distinct paths. Hence, in the following, we use actor simulation rather than emulation.

Information to build the actor twins may be the output of threat intelligence (Barnum 2012; Brown et al. 2015; Tounsi and Rais 2018) or the result of worst-case assumptions. As an example, there are theoretical results on the most effective strategies to select an attack and that correspond to a worst-case for the infrastructure. The MITRE Att&ck matrix (Strom et al. 2018) is a fundamental source of information to model the actors because it is a knowledge base of adversary tactics and techniques based on real-world observations. We can recover missing or uncertain information on an actor by emulating distinct actors, each covering a distinct combination of unknown strategies or parameters to explore the space of possible parameters and discover the worst case, i.e., the actor resulting in the larger risk. An automated adversary emulation can emulate many actors to explore at a low cost and in a detailed way the space of missing information.

When compared against well-established solutions based upon the Hazop and the Chazop methodologies, a twin-based automated adversary emulation is focused not on hazards but on the behavior of a malicious actor that discovers and exploits

vulnerabilities to reach its goals. The proposed emulation requires a larger amount of information on the actor, but it returns more detailed information on how it can reach its goals (Cormier and Ng 2020; Dunjó et al. 2010).

4.3 Running Multiple Adversary Simulations

The distinguishing advantage of the Haruspex platform is the ability to run multiple simulations to discover all the actor paths with any confidence level and without disturbing the infrastructure. The simulations return a large amount of information on each actor, such as the information it can collect, the attacks it selects, and the modules it targets. Further information concerns the probability the actor executes a path and the path success probability. This information can be used to improve the robustness and the resilience of the infrastructure.

It can be formally proved that the emulation of the Haruspex platform never returns a false positive i.e., an attack path that does not exist, provided that all its input vulnerabilities affect the infrastructure modules. The probability of a false negative that is the probability the simulations do not discover a path, decreases with their number. A larger number of simulations improves the confidence level in path discovery and decreases the probability of a false negative. Our experiments confirm that in not trivial infrastructures more than one hundred thousand emulations may be required to reach a confidence level larger than 99% of discovering all the paths.

Both the infrastructure twin and the actor ones can be built from specifications in the design documentation before building an infrastructure. This supports adversary emulation against an infrastructure still to be deployed to predict its robustness and resilience and to increase them using the simulation outputs. This satisfies the security-by-design requirement of some European legislations (Tankard 2016).

4.4 Discovering Any Actor Path

Multiple simulations cannot discover all the paths the remediation policy should stop even when no false-negative exists because the simulations of an adaptive actor only discover the paths it implements in the current infrastructure status, i.e., given the current vulnerabilities and the attacks they enable. Adaptive actors are intelligent, and they will adopt a minimal effort approach. Hence, they will not change their current paths if they are the most effective ones to achieve their goals. Instead, an adaptive actor may react to patch deployment by selecting some paths it has previously neglected due to their low convenience. In brief, an adaptive actor may react by implementing second-choice paths when patches stop its first-choice ones. Suppose as an example, that at some point an actor A can choose one of A_1 and A_2, two attacks with the same postcondition but where the success probability of A_1 is much larger than the one of A_2. If the selection strategy of A privileges success probability or the cost-performance ratio, A will choose A_1 and neglect A_2. Anyway, when a max–min patch set removes the vulnerability enabling A_1, A will implement the second-choice path where it executes A_2.

The previous discussion shows that to stop all the paths of an actor we have to discover both first and second-choice paths, stop them, and check by further simulations that no further second-choice plans exist. This is repeated till the actor cannot implement further paths. For this reason, the Haruspex platform computes the patches to deploy by iterating three steps:

(a) multiple adversary simulations to discover the paths,
(b) apply the remediation policy to compute the max–min patch set to stop all the paths the iterations have discovered
(c) update of the infrastructure twin to model the deployment of patches.

The iteration stops as soon as a) does not discover further paths and the platform returns the last max–min patch set the policy has computed. The overall accuracy depends upon the confidence level of the simulations in a). A reduction in the number of simulations in each iteration may not affect the overall confidence level because an iteration may discover even first choice paths not previously discovered.

Second choice paths show that approaches to discover and stop paths such as penetration tests may be both misleading and ineffective because they discover, at best, all first choice paths but not second choice ones. Their discovery requires a new test after deploying the patches. The situation is even worse because some second-choice plans have a better success probability than first-choice ones (Baiardi 2019). Hence, patching only the vulnerabilities on the first choice paths a penetration test discovers can result in the paradox of increasing the actor success probability.

4.5 Computing a Patch Schedule

Formally, if V_1 and V_2 are two sets of vulnerabilities and stop(V) denotes the max–min patch set of V, we have that

$$|\text{stop}(V_1 \cup V_2)| \leq |\text{stop}(V_1)| + |\text{stop}(V_2)| (*)$$

This is the reason why the remediation policy considers all the paths previous iterations have discovered. This allows the policy to exploit shared vulnerabilities at best and to minimize the patches to deploy. The policy ranks the patches according to the success probabilities of the paths they block and their impacts. Usually, this ranking is not critical due to the low cardinality of a max–min patch set. As detailed in Sect. 6, experimental data confirm that the average cardinality of the max–min patch set is at most 5% of the overall number of vulnerabilities. This also implies that a small number of false-positive vulnerabilities has a neglectable influence on the number of patches to deploy.

5 Continuous Remediation

The Haruspex platform can support continuous patching by monitoring the infrastructure and the infrastructure environments to apply the remediation policy and close the vulnerability window. First, we discuss the changes in the infrastructure and its environment and then continuous patching.

5.1 Evolution of an Infrastructure and its Environment

An infrastructure evolves by integrating new hardware and software modules or by deploying newer module versions. Changes may also affect routing or firewall rules. Active and passive fingerprinting sensors can discover these changes and feed the information to the Haruspex platform. As an example, an active sensor can scan the infrastructure at a predefined frequency to discover new nodes and new modules running on old nodes. Further active sensors may discover updates to the logical interconnection structure or the physical one. A passive sensor sniffs message traffic to discover updates to old modules and new ones. Here an address in an IP packet may reveal the connection of a new node or the creation of a new routing path due to new routing rules.

Further changes are due to the infrastructure environment because of new vulnerabilities or new threat actors. The platform acquires this information from threat intelligence as well as by monitoring some public and/or proprietary databases. The probability that a specific vulnerability is discovered and becomes public is almost unpredictable as it is strongly correlated with the module the vulnerability affects. As a confirmation, Fig. 1 shows the overall number of vulnerabilities discovered in the last twenty years according to www.cvecredentials.com.

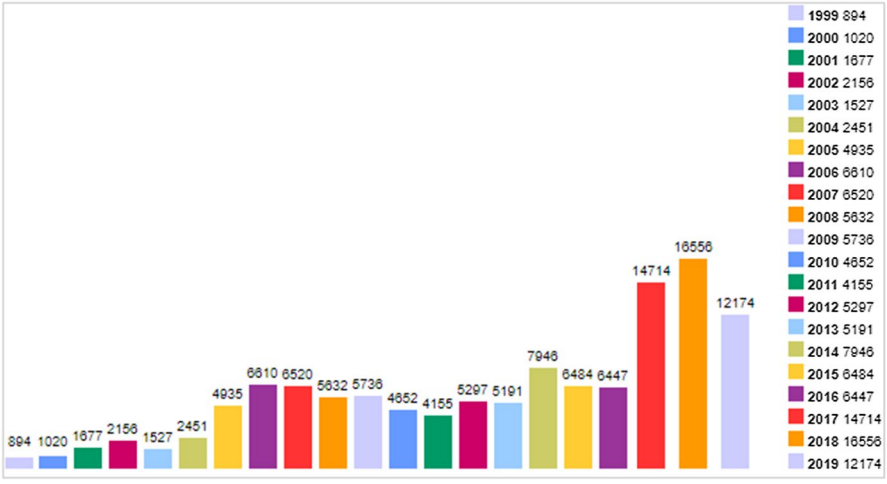

Fig. 1 Number of vulnerabilities discovered per year

Even the number of vulnerabilities discovered in a single module strongly changes. Consider that for Windows 10 this number jumps from 257 in 2018 to 357 in 2019. A similar jump happens for Windows Server 2016 in the same time frame. The number of Debian Linux vulnerabilities discovered in 2018 is about three times those discovered in the previous year.

The previous data and the large differences among the attacks a vulnerability enables imply that a solution cannot predict when some vulnerabilities become public to deploy some countermeasures (Leverett et al. 2012). As discussed in the following, any decision about the adoption of countermeasures should be fired by the discovery of actual changes according to the risk the owner tolerates.

5.2 Discovering and Stopping New Paths

The Haruspex platform uses information from both the sensors and the vulnerability databases to update the infrastructure and the actor twins and to fire simulations. When the simulations discover some new paths, the remediation policy can compute a max–min patch set immediately or delay the computation, and hence the patching, after the discovery of further vulnerabilities. Immediate patching closes the vulnerability window, while a delayed computation of just one max–min patch set may result in a lower patching overhead than when deploying the sets of patches returned by multiple invocations of the policy due to property (*) in Sect. 4.4. A further reason in favor of delayed patching is that experimental data confirm that threat actors are work averse and tend to exploit a few vulnerabilities for a long time (Allodi 2021, Allodi 2015). Hence, the patching of those few vulnerabilities that threat actors exploit and are critical for an infrastructure strongly reduces the risk for a long time. Informally, work-aversion makes it possible to decrease the patching frequency after stopping current attack paths.

If the owner tolerates a probability TP_{succ} of an actor success, the Haruspex platform can delay the patching of an interval δt if $PS(\delta t)$, the actor success probability in δt is lower than Pt i.e., if

$$PS(\delta t) < TP_{succ}$$

The platform checks this condition through the stress curve (Baiardi 2015) a synthetic evaluation of the infrastructure robustness that it computes through the simulation outputs. The security stress $Str_{S,ag,g}(t)$ of an infrastructure S at t due to a threat actor ag with a goal g is defined as the probability that ag reaches g within t. The shape of $Str_{S,ag,g}(t)$ depends upon properties of S and how these properties interact with the strategies and the preferences of ag. Being a probability distribution, $Str_{S,ag,g}(t)$ is monotone, non-decreasing in $0..t$ and $Str_{S,ag,g}(t)(0) = 0$. The stress due to a set of actors sag is the largest stress due to an actor in sag, i.e.,

$$Str_{S,sag,g}(t) = max\,\{Str_{S,agx,g}(t) : agx \in sag\}$$

$Str_{S,ag,g}(t)$ is the worst case for the success probability of ag at t because it assumes ag starts its attack at 0, while in general ag needs some time to discover

which infrastructures are affected by the new vulnerabilities. Hence, a δt delay in the patching is acceptable anytime that

$$Str_{S,sag,g}(\delta t) < TP_{succ}$$

The platform uses $Str_{S,sag,g}(t)$ to compute the largest delay the owner tolerates. The assumption an actor starts its attack as soon as a vulnerability becomes public fails if the actor has an advantage αt i.e,. it is informed αt units of time before the owner. Now, a delay δt is acceptable if

$$Str_{S,sag,g}(\delta t + \alpha t) < TP_{succ}$$

As expected, the convenience of immediate patching increases with αt and, if it is very large, patching should anticipate the vulnerability discovery. The only solution the owner can apply is the *k patch* policy we previously defined because it stops k attacks in a path and this decreases the probability that a single vulnerability recreates the path. k increases with αt.

6 Experimental Results

We discuss the experimental results of the proposed remediation policy for some infrastructures. The NDAs with the infrastructure owners prevent a full data disclosure, but we believe the presented results are interesting and confirm the effectiveness of the proposed approach.

Table 1 shows the results of applying the proposed policy to ten infrastructures in the last two years. These infrastructures were already deployed and they include, among others, an ICS to control an energy production system, the one a cloud provider, a smart factory, and the battlefield of a cyberwar exercise. For each infrastructure, Table 1 lists:

Table 1 Statistics on the Assessments of Some ICT infrastructures

	Vulns	Web Vulns	Total	Attack Paths	Max Min Patch size	ResRisk (%)
1	2172	27	15,209	7462	33	0
2	851	245	4235	9428	34	0
3	571	0	2619	6734	122	16
4	6467	0	23,144	43,876	3373	16
5	1049	0	1863	5223	111	0
6	162	390	2496	0	0	0
7	12,446	296	88,051	76,437	2185	18
8	949	0	4284	4122	18	0
9	86	0	1028	8129	30	0
10	89	0	1836	6783	21	0

1. the number of distinct non-web-related vulnerabilities,
2. the number of distinct web-related vulnerabilities,
3. the overall number of vulnerabilities, i.e., the sum of the occurrences of the various vulnerabilities,
4. the overall number of attacks paths, including second choice ones,
5. the number of patches to deploy, i.e., the size of the max–min patch set,
6. any residual risk.

The number of threat actors ranges from 6, for the second infrastructure, to 14 in the last one. Actors differ because of their attack surface and of their strategies.

Some residual risk remains in two cases where no patches were available for some vulnerabilities i.e., no reduced path exists, see Sect. 3.1. In these cases, the Haruspex platform suggests alternative countermeasures i.e., a new set of firewall rules or the deployment of a honeypot. In case 6, no patch has been deployed because no attack path exists. Here, an actor can create a path by stealing some credentials, i.e., through a phishing message to one of the administrators. The platform emulates social engineering attacks by describing users and administrators as further infrastructure modules while their vulnerabilities enable phishing attacks. The success probability of these attacks is the one that the user/administrator clicks on the link in the phishing e-mail. Infrastructure 9 and 10 were in an early development stage, so both the numbers of distinct and overall vulnerabilities are low. The small size of the max–min patch set for almost all the infrastructures in Table 1 confirms the effectiveness of the proposed policy.

An analysis of the number of the public vulnerabilities in these infrastructures after one year shows the overall number of distinct vulnerabilities increases of a value in the range from three hundred to one thousand. This results in a high number of patches to deploy even when adopting a path-based solution. Instead, a continuous assessment spreads this cost among simpler deployments.

7 Conclusion

We have presented a policy that exploits the knowledge of the attack paths of threat actors to minimize the patching overhead. The Haruspex platform can discover the information the policy needs through multiple adversary simulations that use the digital twin of the target infrastructure and those of the actors. The twin detail level makes it possible to run a large number of simulations in a short time to discover all the paths with a large confidence level.

Experimental results confirm the large reduction in the number of patches to deploy that allows the owner to minimize the vulnerability window of the infrastructure. These results also confirm the effectiveness of continuous remediation to spread the patching overhead across distinct deployments.

Future work concerns the extension of the number of strategies the platform supports to cover a larger number of actors. Furthermore, we plan to model the

occurrence of faults in some modules to use the output of the simulation to handle both malicious actors and random faults.

References

Allodi L (2015) The heavy tails of vulnerability exploitation. In: Piessens FC, Juan, Bielova N (eds) Proceeding of Engineering Secure Software and Systems. Springer

Allodi L, Massacci F, Williams J (2021) The work-averse cyberattacker model: theory and evidence from two million attack signatures. Risk Anal

Applebaum A, Miller D, Strom B, Korban C, Wolf R (2016) Intelligent, automated red team emulation. In: Proceedings of the 32nd annual conference on computer security. ACM

Applebaum A, Miller D, Strom B, Foster H, Thomas C (2017) Analysis of automated adversary emulation techniques. In: Proceedings of the Summer simulation multi-conference. Soc. for Computer Simulation Int., p 16

Baiardi F, Sgandurra D (2013) Assessing ICT risk through a Monte Carlo method. Environ Syst Decision 33:1–14

Baiardi F, Corò F, Tonelli F, Guidi L (2013) Gvscan: scanning networks for global vulnerabilities. In: First international workshop on emerging cyberthreats and countermeasures, Regensburg, Germany

Baiardi F (2015) Harùspex: a suite to assess and manage ICT risk by simulating threat agents. In: Proceedings of Esrel 2015

Baiardi F, Tonelli F, Bertolini A (2015) CyVar: extending Var-At-Risk to ICT. In: Third international workshop on risk assessment and risk-driven testing - volume 9488. Springer, Berlin, Heidelberg

Baiardi F (2019) Avoiding the weaknesses of a penetration test. Computer Fraud & Security 4:11–15

Barnum S (2012) Standardizing cyber threat intelligence information with the structured threat information expression (stix). Mitre Corporation 11:1–22

Brown S, Gommers J, Serrano O (2015) From cybersecurity information sharing to threat management. In: Proceedings of the 2nd ACM workshop on information sharing and collaborative security, ACM, pp 43–49

Chuvakin A, Barros A (2018) Utilizing breach and attack simulation tools to test and improve security, Gartner

Cormier A, Ng C (2020) Integrating cybersecurity in hazard and risk analyses. J Loss Prevent Process Industr 64:104044

Dey D, Lahiri A, Zhang G (2015) Optimal policies for security patch management. INFORMS J Comput 27(3):462–477

Dunjó J, Fthenakis V, Vílchez JA, Arnaldos J (2010) Hazard and operability (HAZOP) analysis A literature review. J Hazard Mater 173(1–3):19

Eckhart M, Ekelhart A (2019) Digital twins for cyber-physical systems security: state of the art and outlook in security and quality in cyber-physical systems engineering. In: Biffl S, Eckhart M, Lüder A, Weippl E (eds) Security and quality in cyber-physical systems engineering. Springer, Cham

Leverett E, Rhode M, Wedgbury A (2012) Vulnerability Forecasting: in theory and practice. arXiv:2012.03814

Manzuik S, Pfeil K, Gold A (2006) Network security assessment: from vulnerability to patch. Elsevier

Martin B (2019) Common Vulnerabilities Enumeration (CVE), Common Weakness Enumeration (CWE), and Common Quality Enumeration (CQE): attempting to systematically catalog the safety and security challenges for modern, networked, software-intensive systems. ACM SIGAda Ada Lett 38(2):9–42

Mell P, Scarfone K, Romanosky S (2006) Common vulnerability scoring system. IEEE Secur Priv 4(6):85–89

Mell P, Kent KA, Romanosky S (2007) The common vulnerability scoring system (CVSS) and its applicability to federal agency systems. US Department of Commerce, National Inst. of Standards and Technology

Mell P, Shook J, Harang R (2016) Measuring and improving the effectiveness of defense-in-depth postures. In: Proceedings of the 2nd annual industrial control system security workshop (ICSS '16). ACM, New York, NY, USA, pp 15–22

Moskal S et al (2018) Cyber threat assessment via attack scenario simulation using an integrated adversary and network modeling approach. J Defense Model Simul 15(1):13

Roytman M, Jacobs J (2019) The complexity of prioritizing after patching. Netw Secur 2019(7):6–9

Sarraute C, Richarte G, Lucángeli Obes J (2011) An algorithm to find optimal attack paths in nondeterministic scenarios. In: Proceedings of the 4th ACM workshop on security and AI, ACM, New York, NY, USA

Strom BE et al (2018) MITRE ATT&CK™: design and philosophy. Technical report

Tankard C (2016) What the GDPR means for businesses. Netw Secur 2016(6):5–8

Tao F, Zhang H, Liu A, Nee AYC (2019) Digital twin in industry: State-of-the-Art. IEEE Trans Industr Inf 15(4):2405–2415

Theoharidou M, Tsalis N, Gritzalis D (2013) In cloud we trust: Risk-Assessment-as-a-Service. In: IFIP international conference on trust management. Springer, Berlin, pp 100–110

Tounsi W, Rais H (2018) A survey on technical threat intelligence in the age of sophisticated cyber-attacks. Comput Secur 72:212–233

Tripathi A, Singh UK (2011) On prioritization of vulnerability categories based on CVSS scores. In: 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT, IEEE)

Wang L, Jajodia S, Singhal A, Cheng P, Noel S (2014) K-zero day safety: a network security metric for measuring the risk of unknown vulnerabilities. IEEE Trans Dependable Sec Comput 111:302014