Dynamic System Diversification for Securing Cloud-based IoT Subnetworks

Hussain M. J. Almohri* Layne T. Watson† David Evans‡
Stephen Billups§

Abstract

Abstract

Remote exploitation attacks use software vulnerabilities to penetrate through a network of Internet of Things (IoT) devices. This work addresses defending against remote exploitation attacks on vulnerable IoT devices. As an attack mitigation strategy, we assume it is not possible to fix all the vulnerabilities and propose to diversify the open-source software used to manage IoT devices. Our approach is to deploy dynamic cloud-based virtual machine proxies for physical IoT devices. Our architecture leverages virtual machine proxies with a diverse set of software configurations to mitigate vulnerable and static software configurations on physical devices. We develop an algorithm for selecting new configurations based on network anomaly detection signals to learn vulnerable software configurations on IoT devices, automatically shifting towards more secure configurations. Cloud-based proxy machines mediate requests between application clients and vulnerable IoT devices, facilitating a dynamic diversification system. We report on simulation experiments to evaluate the dynamic system. Two models of powerful adversaries are introduced and simulated against the diversified defense strategy. Our experiments show that a dynamically diversified IoT architecture can be invulnerable to large classes of attacks that would success against a static architecture.

Keywords: Diversity, Adaptive Security, Optimization, Network security

*Department of Computer Science, Kuwait University, Safat, Kuwait. Email: almohri@cs.ku.edu.kw

†Departments of Mathematics, Computer Science, and Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, VA, USA. Email: ltw@cs.vt.edu

‡Department of Computer Science, University of Virginia, Charlottesville, VA, USA. Email: evans@virginia.edu

§Department of Mathematical and Statistical Sciences, University of Colorado Denver, Denver, CO, USA. Email: stephen.billups@ucdenver.edu

1 Introduction

The Internet of Things (IoT) represents a network of interconnected and autonomous devices that interact with each other with little or no user intervention [47]. A typical smart-home IoT device interacts with the user through application clients, providing simple interfaces for operating devices [45]. Similar to personal computers and mobile devices, the software stack in IoT devices can be based on commonly-used operating systems such as Linux [9] or open-source systems such as Amazon's FreeRTOS [53]. These systems satisfy the requirements of IoT devices in terms of device drivers and the usage patterns. Unfortunately, IoT systems can be subject to attacks when using outdated [42] or when a new attack vector emerges. For example, Amazon's FreeRTOS was vulnerable to a remote attack exploiting the TCP/IP component [38]. Another example is the improper boundary check vulnerability found in Contiki-NG, an open-source operating system for IoT [23]. These and other similar vulnerabilities enable lead to remote exploitation of IoT devices. In some cases (such as a recent vulnerability in an SDK implementing UPnP and SSDP protocols [24]) a remote exploitation also allows for arbitrary code execution on the target with root privileges.

This work aims to mitigate remote intrusions that exploit software vulnerabilities in IoT devices by diversifying system configurations. Previous works for securing IoT networks have taken several directions. Preventive measures include access control and program analysis. In particular, leveraging blockchain access control architectures has been suggested in several previous works (e.g., [28, 56, 1, 13, 35, 32, 7, 25, 54]). Other works (such as [19, 20, 17, 51, 16, 18]) developed program analysis techniques to identify and prevent vulnerabilities in the development stage. Hardening the security of an IoT network can also benefit from secure runtime architectures, for example, by installing firewalls [21] or securing the interaction of devices in an IoT platform [8]. Secure architectures complement preventive measures by actively protecting the network. Further, learning from data produced by attackers, anomaly detection systems can aid secure architectures, enhancing the guarantees provided by preventive solutions [61, 3, 29]. Microsoft has introduced a new tool, Microsoft Defender for IoT, which provides behavioral analysis to defend against unauthorized remote access [50].

Our goal is to provide a preventive solution that produces randomized and diversified *device configuration settings* (such as the choice of the operating system) of devices in an IoT network. Our work is concerned with system software that provide services to the application layer. Vulnerabilities resulting from interactions of IoT applications [20] are beyond the scope of this work. We present a diversified IoT architecture that learns configurations which are likely to be more resilient against attacks. Our architecture restricts access to physical IoT devices through a virtual machine proxy. Each virtual machine proxy produces logs of network requests (for example, `HTTP` messages to and from an application client). The data is sent to an anomaly detection system (such as HAWatcher [29]) to produce potential attack signals. Our architecture uses attack signals to update attack likelihood for the configurations used on virtual machines. A continuous process solves a mixed-integer optimization problem to assess the current software configurations and automatically replace vulnerable virtual machines with ones that are less likely to be the subject of attack.

Our hypothesis is that configuration diversity with randomization can thwart attacks that would bypass other security measures in virtual machines that control access to IoT devices. When a vulnerable IoT device is configured to be only accessible through a virtual

machine proxy, the diversified IoT architecture can improve security by diversifying the virtual machine configurations. Although the concept of diversity (such as [41, 43, 14]) has been previously used to secure software, this work is the first to propose an online dynamic reconfiguration of the IoT networks by solving an optimization problem.

The focus of this work is to provide the optimization model and algorithms for selecting device configuration settings, facilitated by virtual device proxies [3]. There are several challenges for producing diversified device configurations to secure an IoT network. First, it is difficult to identify a device setting that plays a significant role in enabling a remote exploitation attack. For example, in a typical Ubuntu Linux web server, more than fifty system services execute. When detecting that a camera is stealthily sending pictures to an unauthorized client, we know that the device was attacked remotely, but we do not know whether the attack was because of an operating system failure, a network layer exploit, or a web service vulnerability. Note that a successful compromise may require a multistage attack that exploits a remote service followed by a local service to escalate privileges. Dynamically diversifying configurations evolve the security of the network over time by quarantining and avoiding specific configuration settings (when possible) used in frequently attacked devices. Thus, the network has the opportunity to learn, at some granularity, that a specific group of device settings are responsible for a higher attack frequency on the target devices.

Second, the IoT network needs a system to collect data, learn vulnerable configurations, and select the least vulnerable configurations to be used in device proxies. Selecting a configuration requires detecting attacks on individual machines and using attack data to determine new configurations. This work does not address machine learning algorithms for analyzing raw network data and detecting an attack attempt. Instead, we assume that a machine learning algorithm actively provides intrusion signals. Previous works such as HAWatcher [29] and the attack resilient architecture [3] can be used to produce attack signals. The signals are supplied as input to an integer optimization problem for learning the least vulnerable configurations. We develop the optimization problem and the system's constraints and give an algorithm for actively monitoring the system's attack reports and applying the necessary configuration changes.

Third, modifying the structure of a device requires designing a system that has remote administrative privileges on each IoT device. When a reconfiguration is needed, the system must remotely modify the selected configurations, often requiring a significant amount of time to transfer and install the new configuration, and perform a device reset. A live device update causes unnecessary downtime and potentially enables more remote attack vectors. We address this challenge by leveraging the proxy-based IoT architecture from [3]. Each IoT device connects to a virtual cloud-based proxy. The access control rules only allow incoming and outgoing connections to and from a proxy device. These rules are simple to apply, and the router that connects the IoT device to the Internet can enforce them. When the network decides to deploy a new configuration, a cloud-based service replaces the proxy that connects the IoT device with a different structure.

The main contributions of this work are the design of a diversified IoT architecture and the algorithms to continuously learn optimized device configuration settings based on collected data from an intrusion detection system. In particular, this work provides:

1. the design of the diversified IoT architecture (Section 4.2),

2. the formulation of an integer optimization problem to produce optimal device configurations using the available attack data (Section 4.4), and

3. a case study to present a proof-of-concept design of a proxy-based IoT device, cloud infrastructure, and client (Section 5).

The security of the proposed method is analyzed through a probabilistic model measuring the success of remote intrusion attacks(Section 6.2). Finally, an empirical evaluation of the efficacy of the device configuration selection approach against various remote attack profiles is presented (Section 7). Our results demonstrate that, after several data re-evaluation iterations, the proposed solution quickly learns a set of optimal configuration settings.

2 Background

This section introduces a typical IoT network that can be vulnerable to remote exploitation attacks, followed by an overview of the proxy-based IoT architecture as introduced in Almohri et al. [3].

2.1 Example IoT Network

A typical IoT subnetwork in a home or office environment includes IoT devices that connect to the Internet through a local router. The devices are often independently accessible through the Internet using client applications designed to operate specific devices. For example, a smart home can include networked lights, kitchen appliances, thermostats, locks, surveillance cameras, and voice-operated devices such as Amazon's Echo. Systems such as If-This-Then-That (IFTTT) provide simple programmable interfaces to extend the functionality of IoT devices [46]. Some IoT devices independently connect to the Internet through a router while others connect to a hub. In this work, we do not focus on vulnerabilities specific to devices that require a hub to connect. The solution presented here targets devices that directly connect to the Internet.

Devices run common operating systems such as general-purpose variants of Linux or specially-designed IoT operating systems such as RIOT OS [6] or Amazon's FreeRTOS [53]. Devices also run application-layer servers to respond to client requests on behalf of the *owner* of the IoT subnetwork (usually a homeowner or an office administrator). A typical setup of a home or small office IoT subnetwork involves direct and independent interaction of various devices with their client applications, usually through a centralized vendor-specific cloud-based server. As the number of devices grows, the complexity of handling security vulnerabilities and remote exploitations increases, requiring special architectural considerations.

2.2 Proxy-based IoT Networks

The diversified IoT architecture reuses the attack-resilient architecture in which IoT devices are accessed through device proxies [3]. The trusted computing base consists of two components: (1) a controller, and (2) a network of device proxies (Figure 1). The controller is

a system that is assumed to be secure against known attacks and regularly monitored for patches and security updates. The controller includes software components and storage for collecting data from the IoT subnetwork. The controller has full access to each IoT device and must always maintain connections with all IoT devices. The controller can manage security policies for the IoT application. The controller also manages firewall rules across the network or for individual IoT devices. The controller can deploy and control device proxies. A device proxy can be installed as a daemon on the network switches or routers. Alternatively, the device proxy can be a virtual machine on a cloud computing platform or a private virtual cloud. We assume that the attack-resilient IoT architecture detects successful attacks using anomaly detection in network data. The diversified IoT architecture utilizes the attack data to adjust configurations for proxy virtual machines.

In the proxy-based IoT network, the defense includes a trusted computing base, the physical IoT devices and their installed software, and the network that connects the IoT devices. The attacking front includes software for remotely collecting reconnaissance data about the target IoT subnetwork. The attacker can search for the IP addresses of the target IoT subnetworks and attempt to collect software fingerprints. The attack software can use the reconnaissance data, locate an available exploit, and launch a remote attack. Each attacker works alone, with no coordination with other attackers. However, the attack software can launch parallel attacks. If the attacker compromises an IoT device and modifies messages sent from the device or redirects messages to an attacker-controlled device, the attack has succeeded. If the attack fails, the attacker attempts a new offense until the attack succeeds.

3 Related Work

Diversity as a fault-tolerance method [41] has been the subject of numerous studies, including N-version programming [43], distributed replication [11], code-level randomization [37], anomaly detection [52, 29, 3, 49], and N-variant systems [22]. The idea of diversity in program design is to systematically diversify the development of computer programs through the entire software development life cycle. This approach results in N versions of a program that execute concurrently, tolerating faults in some of the executing versions. The same idea was sought as a security mechanism against design faults caused by computer viruses [36]. If the program was carefully diversified, the computer virus is unlikely to infect all the concurrently executing N versions. In a denial-of-service scenario, for example, the benefit is that only the infected version becomes unavailable while the healthy versions continue service. In a remote intrusion attack, all versions must be vulnerable and the attacker must have the capability to attack all versions to succeed. Jackson et al. argue that randomization of instruction sets, system calls, and library entry points results in exponential difficulty of exploitation [34].

Designing secure IoT environments can benefit from diversifying the software stack used in individual devices. This diversification scheme is powerful but is not specific to IoT environments. The primary challenge in securing IoT systems is in providing resiliency at the network-level [62]. The present work is the first to propose the design of an IoT environment in which configurations of individual IoT devices are diversified. For example, a diversified choice of operating system can thwart various attack vectors. In a study of 15

years of reported vulnerabilities in 11 operating systems, Garcia et al. reported that a low number of vulnerabilities were shared among the studied operating systems [30].

Cloud-based diversification for improving system quality is a well-established idea. This work extends cloud-based diversification to use attack data. We also show that the diversification approach can be used as a protective layer over a network of physical IoT devices. Guo et al. developed a game-theoretic model to study a reactive system that chooses from several available operating system options to diversify cloud-based virtual machines [31]. CosTLO [60] lowers latency by finding cost-effective configurations for cloud systems. Yuan et al. proposed using a moving target defense to dynamically reconfigure network settings to combat passive eavesdropping [63]. This work shares the idea of dynamic configuration but differs in using dynamic proxy configurations to protect against remote intrusions, and learning from a history of attacks to further improve diversification over time.

More recently, there have been attempts to study and employ diversity for secure cloud storage [10], multicloud security [4], network intrusion [14], and security of infrastructure systems [40]. Zhang et al. developed a metric to study the effectiveness of diversity against classical network attacks [64]. While diversity in distributed environments has been the subject of classical works (such as the work by Birman [11] and ZooKeeper [33]), few works have provided dedicated models for measuring the effectiveness of diversity. A subsequent work suggested system diversity to mitigate attacks against unpatched vulnerabilities [14] and zero-day attacks [59].

4 Diversified IoT Architecture

This section presents the details of the diversified IoT architecture. The critical aspect of the presented architecture is to use a layer of cloud virtual proxy machines that relay network requests to and from IoT devices. The cloud software is modified, leaving the underlying software on physical IoT devices intact. Our architecture uses anomaly detection system signals to label software configurations as vulnerable when a cloud proxy is compromised. An optimization method recomputes a new selection of software configuration deployed on cloud proxy machines to move away from the vulnerable configuration. Note that this work does not address the problem of anomaly detection, assuming an anomaly detection method exists in parallel to aid the diversification process.

The rest of this section presents a threat model concerning remote intrusions (Section 4.1), an overview of the diversified IoT architecture (Section 4.2), the proposed software configuration method (Section 4.3), and the optimization we use to select configurations (Section 4.4).

4.1 Threat Model

We assume that attackers can establish remote connections and have no physical access to the target IoT devices. The attackers know the IP address ranges for the target IoT subnetworks (IP addresses could be found by analyzing client applications that connect to the target devices). Attackers can analyze the configurations of remote machines. Since devices are only accessible through proxy machines, the attackers would only be able to examine the configuration of proxy machines. Attackers do not initially have a complete

and accurate view of the configurations used in the targeted proxy machines. Attackers can perform black-box testing to find the available and exploitable vulnerabilities in the targeted proxy machines.

We assume that the actual IoT devices are highly vulnerable because of executing outdated software for which vulnerabilities have been exposed. Thus, when a proxy machine is compromised, the attacker can exploit the target IoT device. In this work, we also assume that attackers are motivated by objectives beyond mere compromise of the IoT devices. The ultimate attack goal is to produce deceitful messages using the compromised IoT devices. As a result, the attacker aims to divert the activities of the IoT subnetwork towards malicious outcomes. When the attacker gains access to the proxy virtual machine, the attacker can exploit the corresponding IoT machine after spending some time to perform the exploitation.

The defense (in the target network) has an exploitation detection mechanism that may be imprecise and requires a period of data analysis. When the IoT subnetwork suspects that a node u is compromised, a compromise counter for the configuration used in u is incremented to affect the outcome of the upcoming configuration selection as described next. The defense may be imprecise in detecting the compromised devices. That is, at the beginning of each interval of reviewing the status of IoT devices, the defense may not know that some vulnerable devices were actually compromised. That said, we assume that the defense eventually detects if an IoT device is compromised, even though the detection can miss several iterations of attacks.

4.2 Design Overview

The diversified architecture extends the proxy-based architecture by diversifying and dynamically modifying the proxy virtual machines that interface static and vulnerable IoT devices (Figure 1). Remote clients only access IoT devices through a diverse set of proxy virtual machines $(P_1, P_2, \ldots)$ with varying configurations. The configurations evolve over time to learn the attacker's skills and disable the attacker from accessing a compromised proxy configuration in the future. A device selector periodically executes a device selection algorithm (Algorithm 1) and replaces old proxy machines accordingly.

To illustrate what we mean by configuration diversification, Table 1 shows an example diversification of three proxy machines (the rows) for three categories of configuration options (the choice of the operating system, the choice of the web server, and the choice of the transport layer security implementation). The configurations in Table 1 would change over time to move away from vulnerable configurations (if detected to be compromised previously) and to randomize the choice of configurations to avoid predictability. Selecting the least vulnerable configuration is trivial. However, diversifying the configurations for which vulnerability data is incomplete is challenging and requires using integer optimization as described in Section 4.4.

Figure 2 shows the lifecycle of our solution. The IoT network initializes with a random device proxy to tunnel access to and from the target proxy device we want to protect (Step 1). Next, each virtual machine proxy produces network request logs and stores them in a cloud database (Step 2). An anomaly detection algorithm receives the network request logs to produce anomaly signals. Note that this work does not address the specific methodology for anomaly detection and assumes the availability of existing practical algorithms such as

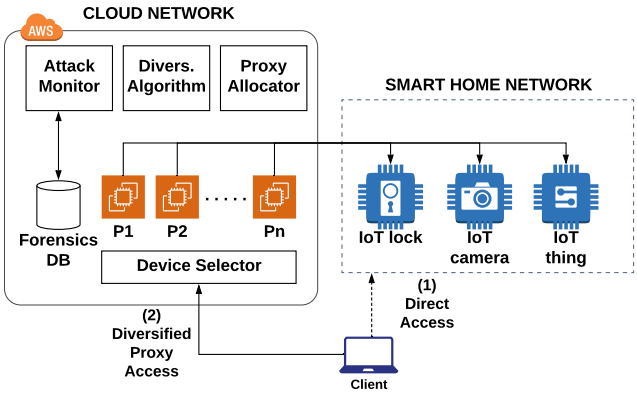


Figure 1: Architecture of cloud-based IoT device diversification. Shadowed subnetworks are not accessible from the Internet. Each IoT device is only accessible through a corresponding proxy machine in the cloud. Attackers can only exploit proxy machines as they respond to Internet requests.

in [3]. The diversification algorithm uses the information provided by intrusion detection to select a new proxy configuration (Step 3). Finally, the proxy allocator deploys the new configuration in a virtual proxy machine (Step 4). The selector requires a set of virtual machine instances corresponding to the various software configuration combinations. The architecture initializes with virtual machines with diverse configurations awaiting deployment in Step 4. At the time of deployment, the allocator selects and turns on the virtual machine that matches the configuration produced by the diversification algorithm. The allocator turns off the current virtual machine and returns it to the pool of available configurations. Note that a virtual machine replaced due to high vulnerability is unlikely to be deployed shortly based on the diversification algorithm.

Our architecture collects network data from the proxy virtual machines to facilitate anomaly detection. Network data comprises requests at the application layer. In our implementation (Section 5), we use `HTTP` requests between client applications that connect to the virtual proxy machines, which in turn relay requests to the physical IoT devices. A record of all `HTTP` requests feed an anomaly detection algorithm that clusters requests to spot unusual patterns. The architecture in Figure 2 uses signals from anomaly detection.

4.3 Diversifying Configuration Selection

This section describes how to select proxy configurations for randomizing and diversifying the attack surface of IoT networks. First, we present a formal model of the Internet of Things and the diversified IoT architecture. The model is then used in Section 4.4 in developing a constrained integer optimization problem for selecting optimal proxy configurations with the objective of increasing diversification (and reducing predictability) and avoiding configurations that are more likely to be compromised by attackers.

The Internet of Things. An IoT subnetwork is a digraph $G_k = (U_k, E_k, L_k)$ with the set U_k of nodes, the set E_k of edges (ordered pairs of distinct nodes), and the set L_k of

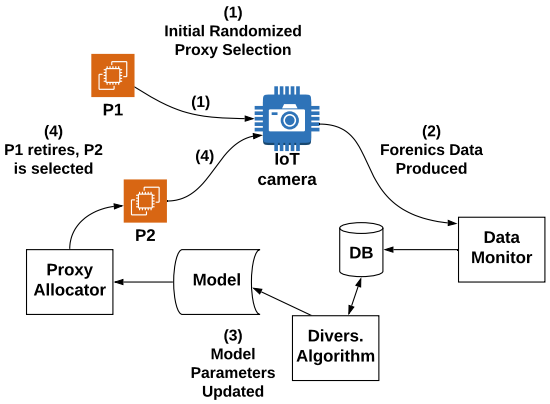


Figure 2: Life cycle of device configuration diversification. The process starts by randomly selecting a proxy P_1 from the available configurations. As the IoT network operates, an attack monitor collects network request data from proxy. The collected data guides selecting a new proxy configuration P_2 if P_1 has a high attack frequency or the algorithm decides to diversify away P_1.

	Operating System			Web Server			TLS		Firewall		
	Win Server	Ubuntu	Amazon Linux	Apache	Nginx	Node.js	Openssl	GnuTLS	R1	R2	R3
v_1	0	1	0	1	0	0	0	1	1	0	0
v_2	0	1	0	0	0	0	0	1	0	1	0
v_3	0	0	1	0	1	0	1	0	0	0	1

Table 1: An example configuration matrix (for devices v_1, v_2, and v_3) that can grow significantly larger. Major software package versions can be added as extra configuration options. The optimization problem is to find a configuration that optimizes a global objective defined by the function $f(X)$ (Section 4.4).

labels for the edges in E_k. k is the subnetwork's universal identifier, distinguishing it from other subnetworks. Two subnetworks G_k and G_l of G are neighbors in G if G has an edge $(u, v) \in (E_k \times E_l) \cup (E_l \times E_k)$. A node in a subnetwork represents an IoT device. Nodes are assumed to be at least connected to one subnetwork, through an Internet protocol. Nodes are not necessarily connected directly to the Internet. Instead, a node can communicate with Internet clients through other nodes. The set of nodes in a network $U_k = V_k \cup W_k$ may contain both proxy nodes V_k and physical nodes W_k.

Attackers may connect to a target IoT subnetwork G_k through several neighboring subnetworks. These neighboring subnetworks are allowed to connect using various protocols such as `HTTP` over `TCP` for the application and transport layers. Attackers can exploit vulnerable protocol implementations (such as [5]) to compromise the devices. Even with a secure communication protocol such as Transport Layer Security (TLS) and QUIC [27], applications on IoT devices can still be vulnerable to remote exploitation attacks.

Diversified IoT Subnetwork. A *diversified* IoT subnetwork $\mathcal{A}$ is an IoT subnetwork that

Symbo	Interpretation
G_k	Digraph representing the network
V_k	Set of proxy nodes
W_k	Set of physical nodes
X	Configuration matrix for all proxies
$\mathcal{Q}$	One configuration vector
$f(X)$	Global objective function
$\rho(X)$	Configuration replacement cost
β	Weight of $\rho(X)$
$\sigma(X)$	Expected vulnerability cost
ω	Weight of $\sigma(X)$

Table 2: A table of major symbols used throughout the article.

(1) has several device proxies connected to every physical IoT device, that is, $|V_{\mathcal{A}}| > |W_{\mathcal{A}}|$, and (2) contains an incomplete bipartite subgraph between the two node subsets $V_{\mathcal{A}}$ and $W_{\mathcal{A}}$, with edges $E^*_{\mathcal{A}} \subset E_{\mathcal{A}}$ that only connect $V_{\mathcal{A}}$ and $W_{\mathcal{A}}$. Note that the number of device proxies for each physical device can vary. Each device proxy can use a distinct configuration from other device proxies used to represent the same physical device. Ideally, all device proxies in $V_{\mathcal{A}}$ should use distinct configurations.

Refreshing device proxies. The diversification algorithm and the proxy allocator (shown in Figure 1) within the cloud subnetwork N_c manage the refreshing process (Algorithm 1). Let `select(`X, z`)` be a function to uniformly select z elements from a sequence (or set) X. At randomly selected times, the controller node selects a subset V^*_k of z device proxies from V_k for refreshing. The subset with minimal current network activity is selected to avoid disrupting active events and network traffic in the physical IoT devices. Avoiding disruption is platform-dependent. For example, in a cloud-based setting, the controller (Figure 1) can examine the current traffic load of the device proxy. Algorithm 1 continues to produce z configuration options in the vector $\mathcal{Q}$, by solving a multiobjective integer constrained optimization problem (described in Section 4.4). Calling `solve` (X,D) uses BARON [55] (as a generic solver) to solve the optimization problem (as explained later in Section 4.4). X is a matrix of configurations for machines (defined later), and D is the dataset showing the security and usage histories of the current configurations.

Then, machines are produced by calling `produce(`$\mathcal{Q}$`)`, given a configuration vector $\mathcal{Q}$. Finally, each existing machine $x \in V^*_k$ is replaced with a new machine $y \in Y$ by calling `replace (`x`, `y`, `G`)`, modifying the underlying network structure captured by the graph G.

Note that the choice of `select` and `solve` are not influenced by attacker-controlled IoT

Algorithm 1 Select configuration and refresh device proxies.

Require: $G = (V_k \cup W_k, E_k), \mathcal{Q}$
1: $V_k^* \leftarrow$ `select`(V_k, z)
2: $\mathcal{Q}_k^* \leftarrow$ `solve`(X, V_k^*)
3: $Y \leftarrow$ `produce`$(\mathcal{Q}_k^*)$
4: **for** $(x, y) \in (V_k^*, Y)$ **do**
5: `replace`(x, y, G)
6: **end for**
7: **return** G

devices (that were compromised). The solution produces by `solve` can only be predicted when attackers are aware of the vulnerabilities of all devices and whether or not the IoT network has indeed observed successful attacks on the vulnerable configurations. That said, the attacker's view of the defense knowledge is limited. To reduce the predictability of the results, `select` also randomizes the configuration selection.

4.4 Optimizing Configuration Selection

This section develops the optimization problem, as the key contribution of this work, which can be solved using standard integer optimization problem solvers (such as BARON [55]). Algorithm 1 uses the solver to compute the optimal value for the optimization problem in the rest of this section. BARON is a general mixed-integer solver for solving optimization problems. This work does not extend or modify BARON but uses it as part of a novel process for dynamically selecting software configurations.

The cloud controller produces device proxy configurations by solving a multiobjective integer constrained optimization problem. The objectives are to minimize the *configuration replacement cost*, *configuration vulnerability cost*, and the number of *future configuration changes*. As the parameters to optimize represent conflicting goals, the optimization is designated as a multiobjective optimization problem. The first aim is to reduce the number of repeated configuration settings between two consecutive configuration selections (for example, avoid using the same kernel version consecutively). The expected device vulnerability is identical for all configurations when the cloud controller has no historical evidence of device compromise. The configuration vulnerability cost is adjusted once event sequences with fabricated consequences are created from device proxies with a particular configuration. Finally, the cloud controller should aim for fewer configuration changes when possible. More changes can cause delays in the functionality of physical devices as the device proxies are being replaced. Assume that future changes are correlated with configuration vulnerability cost; when a weak security configuration is selected, more reconfiguration is needed.

Points on the Pareto front are found by minimizing a convex combination of the individual objectives,

$$f(X) = \beta\rho(X) + \omega\sigma(X),$$

where $\beta \geq 0$, $\omega \geq 0$, $\beta + \omega = 1$, X is a configuration selection matrix (Table 1), $\rho(X)$ is the *configuration replacement cost* of X (a measure of the frequency with which X is changed),

and $\sigma(X)$ is the *configuration vulnerability cost* of X (a measure of the probability of X being compromised).

Each configuration option is represented as a binary decision variable X_{ij}. The ith row of X corresponds to the configuration of the device proxy i, and the jth column corresponds to the configuration option j. The choice of binary variables is because configurations are represented as selections. For example, $X_{12} = 1$ may indicate that for virtual machine 1, the selected operating system (represented by column 2) is Ubuntu Server 18.

Algorithm 1 is executed when the IoT subnetwork has to refresh the configuration of device proxies. After the last device proxy configuration refresh, the IoT subnetwork prepares the values of ρ and σ. Only at the beginning of a new interval δ, the function $f(X)$ is computed to reflect the new computed values for ρ and σ. The configuration replacement cost is computed based on the configuration selections. When a configuration option is selected and deployed, the option is recorded along with a selection counter showing the number of times the configuration was selected for a period of time. The counters are updated when the selected configurations are deployed. Let a_j denote the number of times the configuration j was selected. The function $\rho(X)$ computes the cost of repetition as:

$$\rho(X) = \sum_{ij} a_j X_{ij}.$$

The configuration vulnerability cost is relative to the number of times fabricated messages originated from a set of configuration selections during δ. Let b_j denote the number of times a proxy device with configuration j was reported to generate fabricated messages during $\delta - t$. Then, $\sigma(X)$ computes the cost of vulnerable configurations as

$$\sigma(X) = \sum_{ij} b_j X_{ij}.$$

There are several constraints to the optimization problem. Modeling monetary costs has previously been heavily studied and is not discussed in this work. Thus, we assume all configurations have the same cost. An upper-bound constraint limits the number of device proxies in the system (i.e., the number of rows in X). The combination of columns in a single device proxy is also constrained. For example, suppose that there are two operating system options, Ubuntu Linux 18 and Amazon Linux, represented in columns $j = 1$ and $j = 2$, respectively. No device proxy i can have $X_{i1} = 1$ and $X_{i2} = 1$ at the same time. Thus, the constraint $X_{i1} + X_{i2} = 1$ limits the choice of the operating system.

The solution must also exhibit diversity among the device proxies. For example, no two device proxies for the same physical device should be given the same system configuration at the same time, illustrated by the constraint

$$\sum_{j=1}^{C} \left(X_{1j} - X_{2j}\right)^2 \geq 1,$$

if X were a $M \times C$ matrix and device proxies 1 and 2 were assigned to the same device. The diversity constraint can be applied to all virtual devices for all physical devices such that a degree of diversity is achieved. Diversity in each category of configuration options (such

as the choice of the operating system) is ensured by constraints on the columns. If X is a $M \times C$ matrix and D is the desired diversity such that no more than $n = M - D$ device proxies share option j, then the constraint is

$$\sum_{i=1}^{M} X_{ij} \leq n.$$

This constraint is needed to ensure that the optimization algorithm (the function `solve`) deliberately diversifies configuration selections across various virtual proxy machines. Without the diversity constraint above, `solve` may select the same operating system version for all virtual machine proxies as long as the selected operating system version minimizes the objective function f.

5 Case Study

The presented diversification architecture (Section 4) relies on three components. One is the use of cloud-based proxy machines to access IoT devices. This section reports on a proof of concept implementation of the proxy-based access of IoT devices. Second, the core of the diversification architecture is the optimization algorithm that utilizes the available data to select device configuration settings that are likely to improve the network's overall security. Section 7 describes simulations to evaluate the efficacy of the diversification algorithms. Third, the architecture depends on intrusion detection and data analysis methods to decide if a particular machine has been the victim of attacks. Intrusion detection for IoT devices is beyond the scope of this work. In the remainder of this section, the design and implementation of a proxy-based IoT camera are presented. Finally, an experiment shows the impact of redirecting requests through a proxy on the network performance.

5.1 Design and Implementation

We developed a customized IoT camera to demonstrate the feasibility of accessing IoT devices through a proxy. The IoT camera is constructed based on the design of Figure 3.

Basic Design. The IoT camera responds to requests from a client application to retrieve images. Also, if a motion is detected, the camera sends motion images to the client application. The camera uses a web server as middleware to send and receive image files. The camera is built using a Raspberry Pi 4 camera module (referred to as the module) with a wireless network interface. An original ARM Ubuntu 18 is installed on the module. Images are captured using the open-source `motion` package.

Diversified Design. To diversify the interface with the camera, the client's direct interaction with the camera is replaced by a mediating load balancer in Amazon Web Services (AWS) cloud. The load balancer fixes the IP address for the communication and abstracts the dynamically changing virtual machine proxy. A pool of virtual machine proxy servers is created, each proxy using a different operating system (Ubuntu Server 20.04 LTS, Amazon Linux 2 AMI, SUSE Linux Enterprise Server 15). Each operating system uses a separate build for the Apache web server and transport layer security (TLS). The load balancer uses

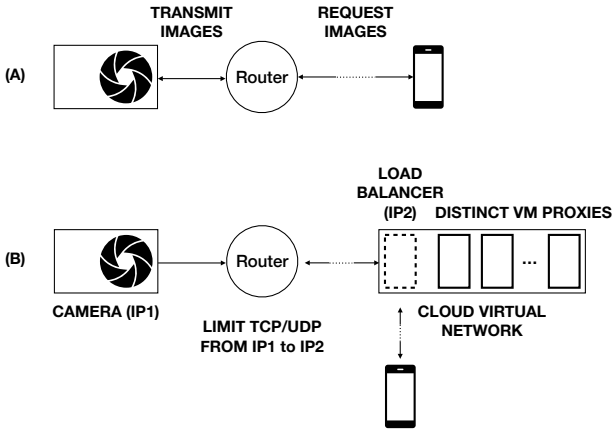


Figure 3: A case study to design a proxy-based IoT camera. Part A shows a direct communication from an IoT camera behind a router with mobile client applications. Part B shows our alternative design using a Raspberry Pi 4 camera module, a restricted firewall rule to limit communication with a cloud load balancer, and a pool of virtual machine instances (in an Amazon Web Services cloud), each with a different operating system configuration to use as proxy machines.

the default round-robin algorithm to distribute the load among the available proxy machines. A listener rule is added for each proxy with an initial equal priority (a priority is an integer value indicating which rule should be followed first). The priority values are modified to forward requests to less vulnerable proxy machines.

5.2 Network Performance

This section presents the results of an experiment to study the impact of the diversified architecture (Section 4) on network performance. First, we measure performance by comparing the speed of a *direct* data transfer with that of a proxy-based data transfer.

Experiment Setup. Two identical Raspberry Pi modules (M_1 and M_2) were used to detect motion and take images using the onboard Camera Module V2 (Figure 3). The two modules are used to measure the network performance of the diversified IoT architecture. M_1 directly connects to a client using a LAN, but M_2 is set to use the diversified architecture. M_1 uses a web server to respond to periodic update requests from the client to retrieve motion images. M_2, on the other hand, connects to the cloud and uploads each image I_k through the load balancer to a designated proxy P_i. The client would send the periodic update requests through a cloud-based load balancer to a proxy virtual machine P_j deployed by Algorithm 1 (Section 4.3). For M_2, the image is stored on a shared Amazon Simple Storage Service (S3) bucket, which is a static cloud-based file system. Regardless of the designated proxy used by the IoT device or the client (even when $i \neq j$), the client can retrieve the corresponding image I_k.

Results. The two modules M_1 and M_2 capture motion images from an identical angle and are set to upload the images upon receiving a request immediately. To measure the *impact of proxy-based image transfer on network performance*, we let the two modules respond to

Method	No. of images	Avg. Transfer Time (MS)	Std. Deviation (MS)	Median (MS)
From cloud	82	502	329	433
To cloud	82	504	327	483
LAN	82	32	20	26

Table 3: Time for data transfer to and from a cloud-based proxy, compared to a local area network direct transfer of images from an IoT device. The average file size for each image is 11 KB. Time values are in milliseconds. The transfer method is from or to cloud, or a direct transfer through a local area network (LAN).

image requests from the client. For each module, a separate client process continuously sends an image request every five seconds. The module uses the camera API to constantly take images and stores them locally (for M_1) or on the S3 bucket (for M_2). This design decouples the requests from image captures, which assists the proxy-based diversified architecture in improving performance.

The average time to transfer images to the cloud-based proxy is approximately 0.5 seconds. Similarly, the average time to move images from the cloud-based proxy to the client is around 0.5 seconds (Table 3). We compared these results with the less likely case that the client and the IoT devices directly communicate over a local area network. As expected, direct communication is significantly faster. However, given that most services require cloud intervention, the network performance of our prototype is reasonable. Note that during the experiments, no other significant network activity was in place.

6 Security Analysis

To analyze the security of the presented solutions, we highlight the achieved security properties and develop a probabilistic model (Section 6.2) for establishing the security boundaries and limitations of the proposed approach. The diversified IoT architecture provides:

1. dynamic resilience against previously undiscovered vulnerabilities (such as `Ripple20` [48], which affects millions of exposed IoT devices) on the platform level, and

2. isolates devices from direct interaction with client applications, shifting the attack surface to dynamic virtual proxy machines.

In this section, we analyze the diversified IoT architecture and show that the achieved security properties are subject to:

1. attacker skills and resources,

2. the availability of remote exploitation vulnerabilities, and

3. the precision of the output of the attack detection algorithms based on which Algorithm 1 selects new configurations.

6.1 Targeted Vulnerabilities

In this work, we address securing devices that directly interact with remote clients. This interaction can result from a design choice (for example, to improve network performance) or the side-effect of using third-party software packages. The side-effect is that the utilized third-party software packages may leave network services open to remote clients. Further, we distinguish between two types of vulnerabilities. Our solution protects against remote exploitation attacks on the *generic* system services used by IoT devices such as web servers or the operating system kernel. For example, a previous version of Amazon's FreeRTOS [53] was vulnerable to arbitrary remote code execution [38] due to a memory corruption exploit. Independent of specific vendor-supplied software, attackers can exploit any IoT devices with a vulnerable installation of FreeRTOS and gain unauthorized remote access. The presented solution does not particularly address vulnerabilities in vendor-specific software (further discussed in Section 6.3).

6.2 Achieved Security

Here, we theoretically model the achieved security of the diversified IoT architecture subject to the targeted vulnerabilities.

Considering remote intrusion attacks on the software platform of an IoT device as Bernoulli trials, let $X_k = 0, 1$ be a Bernoulli variable indicating failure, success for attack event k. Also, let ϕ be the sample space of all possible outcomes, E be the set of events, $\mathcal{P}$ the *attack* success probability distribution, and $\mathcal{Q}$ the *defense* success probability distribution. Represent attack event k as a Markov chain of Bernoulli variables (each corresponding to an attack step) Y_1, Y_2, ..., Y_n such that

$$X_k = \prod_{i \in I} Y_i,$$

where $I \subset \{1, \ldots, n\}$ captures the necessary steps Y_i for $X_k = 1$.

The diversified IoT architecture depends on intrusion detection data. As described in Section 4.2, once an intrusion is detected, the expected vulnerability score of a virtual proxy is increased. Thus, the probability that a defense succeeds partially depends on the accuracy of intrusion detection. In general, the probability that a particular attack event $e_k \in E$ succeeds ($X_k = 1$) depends on the probability of the event η that the attack is *executed* correctly given (i) the posterior probability that the intrusion detection mechanism at the defense side is accurate (represented by Bernoulli variable $B_k = 1$), (ii) the probability that the defense modifies the proxy configuration immediately before the necessary condition for an attack to succeed (captured by the Bernoulli variable $C_k = 1$), and (iii) the probability that Algorithm 1 diversifies away *all* vulnerable configurations for which the attacker is

capable of compromise (captured by the Bernoulli variable $D_k = 1$). Thus, the probability that an attack incident (one attack attempt) succeeds is defined as

$$P[X_k = 1] = P[\eta \mid B_k = 1,\ C_k = 1,\ D_k = 1].$$

Note that defense modification of proxy configurations either follows a predictable procedure (for example, after t time ticks) or randomly selected points based on a particular distribution.

6.3 Limitations

The presented diversified IoT architecture increases the resilience of a smart-home IoT network. That said, the architecture is limited to platform-level diversification of standard open-source software used to set up IoT devices. Diversifying device-specific (or vendor-specific) systems requires changes to the presented architecture to enable diversity for specific software components. For example, vulnerabilities that target application logic on an IoT device cannot be mitigated using the proposed architecture in this work. Instead, a more modular architectural design for IoT devices is needed. Further, open-source components must be designed to enable diversity at specific features of the application logic. One example is to develop a variety of solutions for user authentication with a unified software interface. Then, a diversified security architecture can leverage these components by relinking them at runtime.

A second limitation of the proposed work is the additional components and costs associated with virtual proxy machines. That said, our design does not need to be deployed on a cloud architecture. Instead, an internal network can use software-defined networking [39] and virtualization platforms [2] to achieve a similar result.

7 Efficacy of Diversification

We measure the efficacy of the diversified proxy-based IoT architecture through simulated attacks. Each simulation examines a distinct attack or defense behavior as explained in Section 7.2. The general research question is *whether the proposed diversified approach can effectively protect the target IoT network by isolating vulnerable devices?* In Section 7.1, we present an attack algorithm that is used as the basic attack approach. Algorithm 2 represents an extreme attack scheme that targets every IoT device configuration in repeated attack attempts. The attack results presented in Section 7.2 are produced by slight variations of Algorithm 2.

The code for producing the optimization problem and performing the simulation is published as an open-source project: `https://github.com/kussl/Diverse-IoT`.

7.1 Attack Algorithm

In this section, we describe a strong attack strategy that repeatedly attacks targets, aiming to compromise devices, collect data, and optimize the attack in consequent attack iterations.

Attacks are assumed to occur as *campaigns* (commonly referred to as cyber espionage campaigns [58]). An attack campaign comprises several attack iterations that terminate when the attacker is no longer interested in (or capable of) continuing the campaign. This work is not concerned with reasons that lead to terminating a campaign.

The steps for the attack campaign are summarized in Algorithm 2. Let `compromise(`u`)` be a function to execute remote exploitation steps to compromise a target machine u, and `canattack(`T`,`X^*`,` S`)` a function that outputs $T^A \subset T$, which includes the machines that could be attacked according to the attacker's skills S. The attacker's skills is a vector of size C with elements corresponding to the columns of X^* (a configuration selection matrix). Each element of the vector S is a binary value indicating the possibility to use the configuration to execute an attack. The function `compromise` can be implemented as scripts that exploit a buffer overflow vulnerability [57], inject shell code, and open a remote connection. `compromise` and `canattack` are related. The result of an attack attempt in `compromise` helps adjusting the data produced by `canattack`. Attackers can use reconnaissance attempts to find targets and decide if they are vulnerable, decide if the vulnerabilities can be exploited, and whether in a previous attempt `compromise` was capable of successful exploitation.

The algorithm starts by updating the attacker's knowledge of the target nodes T and the observed configuration matrix X^*. This could be done by performing reconnaissance on the target network. Next, the attacker analyzes T and X^* to decide if there is a subset T^A that can be attacked. The for-loop iterates over uniformly selected nodes (executed by the function `select`) in T^A. In each iteration, the attacker attempts to compromise the selected nodes. Following the for-loop, the attacker verifies if the minimum number N of compromised devices is achieved. If so, the attacker continues to execute the attack beyond compromising the devices, for example, by deleting data from the devices or propagating through the network to compromise devices that are not directly accessible through outside a local area network (LAN). Here, the focus is on preventing the remote exploitation attack. Further attack goals once a device is compromised is beyond the scope of this work.

7.2 Attack Results

This section describes the design and the result of attack simulations using Algorithm 2 and the defense strategy described in Section 4.3. Simulations show the efficacy of the defense using the expected vulnerability score. The expected vulnerability score is a nondecreasing value associated with each virtual proxy with a distinct software configuration. The defense evaluates the results collected from attacks and increments the expected vulnerability score(by adding one to the score of each device that was found under attack) if the proxy is believed to have been the subject of an attack.

Recall that we assume the defense is using a machine learning algorithm to perform pattern analysis and detect likely attacks. The expected vulnerability cost, as defined in Section 4.4, indirectly measures and positively correlates with the probability of compromise and the cost (such as economic and social cost) of being compromised.

Simulation Setup. The simulator selects a random set of available attacker skills to exploit the configurations in each iteration and sets the corresponding skill values to 1 in the vector S (representing attacker skills, defined in Section 7.1). For each configuration $X_{ij} = 1 = S_j$, the

Algorithm 2 The algorithm for an attack campaign against a target network G_k.

Require: X^*, N, T, S

while attack campaign should continue **do**
  Update T and X^*
  $T^A \leftarrow$ `canattack(`T`,`X^*`,` S`)`.
  for $u \in$ `select(`T^A`)` **do**
    `compromise(`u`)`
    if attack is successful **then**
      add u to the set T^c.
    end if
  end for
  if $|T^c| \geq N$ **then**
    Execute the attack.
  else
    Halt for h time units.
  end if
end while

attacker can exploit a vulnerability in the configuration to enable remote access to the target proxy virtual machine. The diversity parameter (Section 4.4) is set to $D = \lfloor C/M \rfloor \ll M$ (C is the number of configurations and M is the number of proxy nodes). Recall (from Section 2.2) we assume attackers that are capable of compromising the virtual machines can eventually compromise the corresponding IoT devices. The number of nonzero values in S is arbitrarily chosen to be 20% of the number C of available configuration options from which the defense can select a proxy virtual machine. Note that a random choice of attacker skills represents the reality of the variation in tools and exploits available to attackers. It is expected that attackers do not always target identical exploits, hence the assumed uniform distribution of possessed attack skills (matching exploitable targets).

The initialization parameters for each simulation are presented in Table 4.

Defense Assumptions. We assume that the defender can review data from virtual machines using an external controller machine and use anomaly detection to detect malicious messages generated from a specific machine. Two issues concern this assumption. First, the virtual machines emit application messages that are used by the central controller. In a virtual cloud-based network or a software-defined network, these messages can be collected and stored by the trusted central controller. Second, an anomaly detection system that is used by the central controller can *partially* detect malicious application-level traffic from virtual machines. That is, we assume that the anomaly detection algorithm has a nonzero false negative rate. That said, given enough time, the anomaly detection will detect whether a particular virtual machine is compromised as we assume that the attacker generates enough traffic to be analyzed and detected.

Note that even though the defense can analyze traffic generated by virtual machines, we assume the defense cannot detect and block all attackers. The defense has no information on the number of attackers, their origin addresses (that can be spoofed), whether an attack is undergoing, and if a virtual machine that is not actively sending traffic was compromised.

Thus, the diversity and randomness introduced in this work are useful to protect against obscure attack attempts.

Simulation	Number of Configs	Vulnerable Configs	Detection Accuracy	β	ω
A	32	20%	100%	0.1, 0.5, 0.9	0.1, 0.5, 0.9
B	32	20%	20%, 45%, 70%, 95%	0.5	0.5
C	16, 32, 64, 128	20%	95%	0.5	0.5

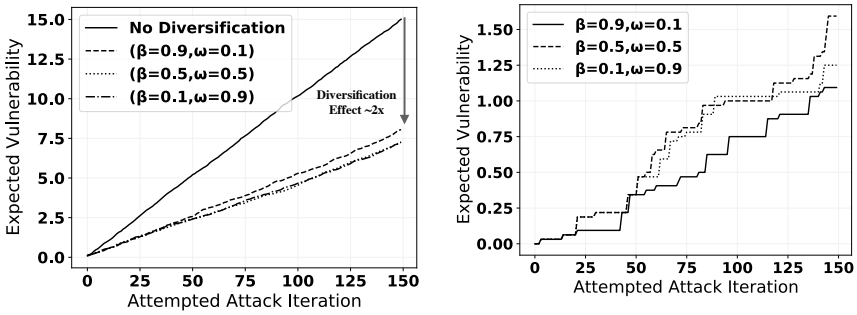
Table 4: Initialization of simulation parameters. β is the weight for the expected proxy replacement cost and ω is the weight for the expected vulnerability cost for optimizing the configuration selection (Section 4.4). For all settings, we use 8 devices, Diversity parameter 4, and 1 proxy per device.

Measuring Success. We measure the attacker's success by the expected vulnerability cost (which is a value to be minimized by the defense; see Section 4.4). The attacker succeeds when attacking a machine with a vulnerable configuration for which the attacker has the skills. For simplicity, in the simulations, each configuration represents a remote service that, if compromised, allows the attacker to control the machine. As the attacker succeeds in compromising more machines, the expected vulnerability cost increases. Thus, the attacker's success is a real-valued quantity as opposed to a binary result. In each iteration of the simulation, the defense reviews the attack data. If an attack attempt was successful in the previous iteration, the defense updates the security configurations used in the compromised proxy virtual machine. The configuration vulnerability cost for each configuration used in the compromised proxy is incremented by one. The configuration replacement cost for each selected configuration is also incremented by one at the beginning of each simulation iteration.

The expected vulnerability cost accumulates through the iterations. The experiment is repeated with various values for β and ω to evaluate the effect of the two components on the objective function. Note that in this scenario, the defense adjusts the configuration in each iteration, assuming the defense has an active reconfiguration policy. The expected vulnerability score stabilizes after about 100 iterations.

Configuration Choices. The results of Figures 4a and 4b show 150 iterations of attack attempts with 32 available configurations and eight IoT devices connected through eight proxy virtual machines. Recall that a software configuration for a proxy only diversifies the platform (such as choosing operating system, TLS, web server). A software configuration can be formed from numerous available choices. For example, the Linux operating system has several distributions (e.g., Ubuntu, Debian, Red Hat), each available in a variety of supported versions (it is possible for an older version to be free of a vulnerability introduced in a newer version). Further, one can choose from several available open source TLS implementations (e.g., GnuTLS [44], Tink [26], OpenSSL [15], Rustls [12]).

Simulation A. In this simulation, we study the effect of two different attacker behaviors on the security of a diversified IoT network: (i) an aggressive and comprehensive attack campaign, and (ii) a step-wise campaign. The simulation for each scenario is repeated three



(a) Aggressive attack (b) Step-wise attack

Figure 4: 150 iterations of an attack campaign in which the defense modifies the configuration setting after each attack attempt. y-axis shows the cumulative expected vulnerability cost for all proxy machines and x-axis indicates the number of iterations increasing in time. The expected vulnerability is a cumulative measure of the frequency of attacks on all proxy machines used by the defense. It grows as more attacks are observed on machines.

times with three different parameter settings to also measure the effect of the two major hyperparameters. β refers to the weight of configuration replacement cost and ω refers to the weight of the expected vulnerability cost (Section 4.4). In each attack iteration of the aggressive campaign, *all* targets that are vulnerable for which the attacker has the required tools and skills are exploited. That is, the for-loop in Algorithm 2 is fully explored. Figure 4a shows the results of the aggressive attack scenario. Note that the defense reconfigures the system in each step. In the second attack scenario, the attacker only uses a single vulnerable target and does not reveal the intent to exploit other targets. To see if this attack campaign strategy is effective, we executed this strategy and produced the results of Figure 4b. This scenario captures time and skill limitations of the attack campaign. As the results show, the attacker is less successful as the overall expected vulnerability achieved is lower.

One notable result is by comparing the line showing the case where no diversification was in place. In this experiment, the same attack sequence is used by the defense only picks one set of configurations (randomly) at the beginning of the simulation and does not change the configurations during the following 149 iterations of attack. Observe that at the last iteration a constant defense has a twice higher expected vulnerability value compared to the diversified architecture, showing the efficacy of the approach under the provided assumptions.

Simulation B. In this simulation, we study the effect of attack detection accuracy on the expected vulnerability score. Precisely, we would like to know the dependency of the accuracy of detection an attack on the diversification method.

In the conducted experiments, we used four different attack detection accuracy values. A detection accuracy controls selecting new configurations in response to attacks. If attack is detected, the data generated from the attack is used to select new configurations.

Simulation C. Finally, Figure 6 shows the effect of the number of available configuration options for a single component in the virtual machine (e.g., the choice of operating system, including various kernel versions). As the number of configurations grows, the expected vulnerability declines, suggesting the difficulty of the attacker in compromising the target

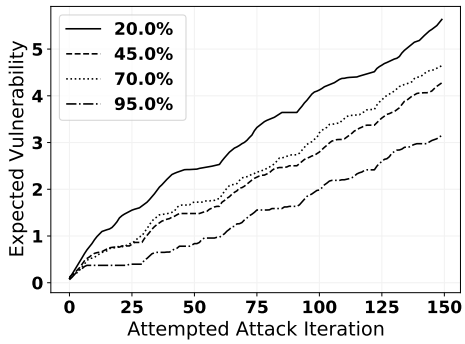


Figure 5: Effect of detection accuracy on isolating vulnerable configurations. Percentages indicate the accuracy of detecting if an attack has occurred following an attack iteration. Higher y-axis values indicate more vulnerable selections.

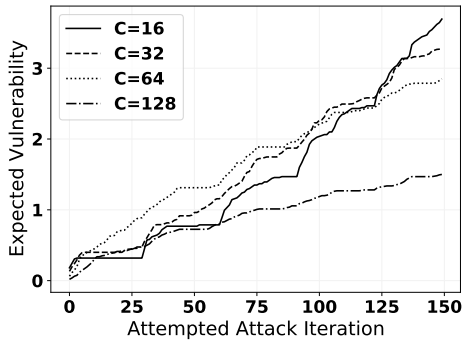


Figure 6: 150 iterations of an attack campaign in which the accuracy of the defense in detecting the attack varies by a probability value $P = 0.9$ with various numbers of configurations (represented by C). A higher number of configurations lowers the expected vulnerability score of the defense.

machines. As expected the earlier iterations show closer values for all different values of C. That said, $C = 128$ shows a steady and slow growth compared to lower values. The core assumption of this simulation is that configurations are chosen such that each is distinct in terms of security vulnerabilities. Also, the available configurations have a smooth distribution of vulnerabilities. That is, there are both weak and strong configurations. Thus, as the number of available configurations grow the expected vulnerability of the network remains lower over time, compared to a small number of configurations.

8 Conclusions

This work demonstrated the use of platform software diversity to mitigate remote intrusion attacks on IoT devices. Diversification relies on optimizing the selection of software configuration settings according to continuously collected anomaly detection signals. We showed the formulation of the problem as an integer optimization problem, which (given accurate

data) can result in a stable state where the most secure configurations are selected. There are three key features of the proposed method. First, one can observe from the simulation results that the defense can quickly discover the *current* best configurations, trained by actual attack incidents. Second, the defense can continue to learn about the state of security in the available configurations. When the system administrator patches vulnerable software or receives more secure updates, the newly updated software results in a new configuration. With the new (and more secure) configuration on a virtual machine proxy, fewer attack incidents will be observed over time, and thus the new configuration will be favored by the optimization problem. Third, system administrators can conveniently implement the diversified IoT architecture using the prominent cloud-computing platforms. As demonstrated in our case study, our approach is feasible and causes predictable network delays due to the extra round-trip to the cloud, which can be a reasonable cost for the provided security level.

References

[1] F. A. Abadi, J. Ellul, and G. Azzopardi. The blockchain of things, beyond bitcoin: A systematic review. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1666–1672, New York, NY, USA, July 2018. IEEE.

[2] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. *ACM Sigplan Notices*, 41(11):2–13, 2006.

[3] Hussain M. J. Almohri, Layne T. Watson, and David Evans. An attack-resilient architecture for the Internet of Things. *IEEE Transactions on Information Forensics and Security*, 15, 2020.

[4] M. A. AlZain, E. Pardede, B. Soh, and J. A. Thom. Cloud computing security: From single to multi-clouds. In *2012 45th Hawaii International Conference on System Sciences*, pages 5490–5499, New York, NY, USA, 2012. IEEE.

[5] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Bias: Bluetooth impersonation attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 549–562, New York, NY, USA, May 2020. IEEE.

[6] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt. Riot os: Towards an os for the internet of things. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 79–80, New York, NY, USA, 2013. IEEE.

[7] L. Bai, M. Hu, M. Liu, and J. Wang. BPIIoT: A light-weighted blockchain-based platform for industrial IoT. *IEEE Access*, 7:58381–58393, 2019.

[8] Musard Balliu, Massimo Merro, and Michele Pasqua. Securing cross-app interactions in iot platforms. In *IEEE Computer Security Foundations Symposium*, pages 319–335, New York, NY, USA, 2019. IEEE.

[9] Edoardo Barbieri. Internet of things and ubuntu: 2021 highlights, December 2021. https://ubuntu.com/blog/iot-and-ubuntu-2021.

[10] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *ACM Trans. Storage*, 9(4), November 2013.

[11] Kenneth P. Birman. Replication and fault-tolerance in the ISIS system. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, SOSP '85, page 79–86, New York, NY, USA, 1985. Association for Computing Machinery.

[12] Joseph Birr-Pixton. Rustls - a modern TLS library, 2022. https://docs.rs/rustls.

[13] S. Biswas, K. Sharif, F. Li, B. Nour, and Y. Wang. A scalable blockchain framework for secure transactions in iot. *IEEE Internet of Things Journal*, 6(3):4650–4659, June 2019.

[14] Daniel Borbor, Lingyu Wang, Sushil Jajodia, and Anoop Singhal. Securing networks against unpatchable and unknown vulnerabilities using heterogeneous hardening options. In Giovanni Livraga and Sencun Zhu, editors, *Data and Applications Security and Privacy XXXI*, pages 509–528, Cham, 2017. Springer International Publishing.

[15] Matt Caswell. Using TLS1.3 With OpenSSL, Feb 2018. https://www.openssl.org/blog/blog/2018/02/08/tlsv1.3/.

[16] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. Sensitive information tracking in commodity IoT. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1687–1704, New York, NY, USA, 2018. IEEE.

[17] Z. Berkay Celik, Earlence Fernandes, Eric Pauley, Gang Tan, and Patrick McDaniel. Program analysis of commodity iot applications for security and privacy: Challenges and opportunities. *ACM Computing Surveys*, 52(4), aug 2019.

[18] Z. Berkay Celik, Earlence Fernandes, Eric Pauley, Gang Tan, and Patrick McDaniel. Program analysis of commodity iot applications for security and privacy: Challenges and opportunities. *ACM Comput. Surv.*, 52(4):74:1–74:30, August 2019.

[19] Z Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated IoT safety and security analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC'18)*, pages 147–158, Berkeley, CA, USA, 2018. USENIX.

[20] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In *NDSS*, Reston, VA, USA, 2019. Internet Society.

[21] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Lannan Luo. Pfirewall: Semantics-aware customizable data flow control for home automation systems, 2019.

[22] Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser. N-variant systems: A secretless framework for security through diversity. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, USA, 2006. USENIX Association.

[23] National Vulnerability Database. Cve-2021-21410 detail, June 2021. https://nvd.nist.gov/vuln/detail/CVE-2021-21410.

[24] National Vulnerability Database. Cve-2021-35393 detail, August 2021. https://nvd.nist.gov/vuln/detail/CVE-2021-35393.

[25] S. Ding, J. Cao, C. Li, K. Fan, and H. Li. A novel attribute-based access control scheme using blockchain for iot. *IEEE Access*, 7:38431–38441, 2019.

[26] Thai Duong. Introducing the Tink cryptographic software library, August 2018. https://security.googleblog.com/2018/08/introducing-tink-cryptographic-software.html.

[27] Lars Eggert. Towards securing the internet of things with QUIC. In *Proceedings of the 2020 Workshop on Decentralized IoT Systems and Security (DISS)*, Reston, VA, USA, 2020. Internet Society.

[28] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke. Blockchain technologies for the internet of things: Research issues and challenges. *IEEE Internet of Things Journal*, 6(2):2188–2204, April 2019.

[29] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. HAWatcher: Semantics-Aware anomaly detection for appified smart homes. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4223–4240, Berkeley, CA, USA, Aug 2021. USENIX Association.

[30] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro. Os diversity for intrusion tolerance: Myth or reality? In *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, pages 383–394, New York, NY, USA, 2011. IEEE.

[31] Minzhe Guo and Prabir Bhattacharya. Diverse virtual replicas for improving intrusion tolerance in cloud. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, CISR '14, page 41–44, New York, NY, USA, 2014. Association for Computing Machinery.

[32] J. Huang, L. Kong, G. Chen, M. Wu, X. Liu, and P. Zeng. Towards secure industrial IoT: Blockchain system with credit-based consensus mechanism. *IEEE Transactions on Industrial Informatics*, 15(6):3680–3689, June 2019.

[33] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, page 11, USA, 2010. USENIX Association.

[34] Todd Jackson, Babak Salamat, Gregor Wagner, Christian Wimmer, and Michael Franz. On the effectiveness of multi-variant program execution for vulnerability detection and prevention. In *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, MetriSec'10, New York, NY, USA, 2010. Association for Computing Machinery.

[35] T. Jiang, H. Fang, and H. Wang. Blockchain-based internet of vehicles: Distributed network architecture and performance analysis. *IEEE Internet of Things Journal*, 6(3):4640–4649, June 2019.

[36] M. K. Joseph and A. Avizienis. A fault tolerance approach to computer viruses. In *Proceedings. 1988 IEEE Symposium on Security and Privacy*, pages 52–58, New York, NY, USA, 1988. IEEE.

[37] Morgon Kanter and Stephen Taylor. Diversity in cloud systems through runtime and compile-time relocation. In *2013 IEEE International Conference on Technologies for Homeland Security (HST)*, pages 396–402, New York, NY, USA, 2013. IEEE.

[38] Ori Karliner. Freertos tcp/ip stack vulnerabilities – the details, Dec 2018. https://blog.zimperium.com/freertos-tcpip-stack-vulnerabilities-details/.

[39] Keith Kirkpatrick. Software-defined networking. *Communications of the ACM*, 56(9):16–19, 2013.

[40] J. Kirsch, S. Goose, Y. Amir, D. Wei, and P. Skare. Survivable scada via intrusion-tolerant replication. *IEEE Transactions on Smart Grid*, 5(1):60–70, 2014.

[41] K. C. Knowlton. A combination hardware-software debugging system. *IEEE Transactions on Computers*, C-17(1):84–86, 1968.

[42] Shebu Varghese Kuriakose. Secure ota updates for cortex-m devices with freertos, July 2021. https://www.freertos.org/2021/07/secure-ota-updates-for-cortex-m-devices-with-freertos.html.

[43] Liming Chen and A. Avizienis. *N*-version Programming: a fault-tolerance approach to reliability of software operation. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'.*, pages 113–120, New York, NY, USA, June 1995. IEEE.

[44] Nikos Mavrogiannopoulos. GnuTLS and TLS 1.3, May 2018. https://nikmav.blogspot.com/2018/05/gnutls-and-tls-13.html.

[45] Yan Meng, Wei Zhang, Haojin Zhu, and Xuemin Sherman Shen. Securing Consumer IoT in the Smart Home: Architecture, Challenges, and Countermeasures. *IEEE Wireless Communications*, 25(6):53–59, 2018.

[46] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. An empirical characterization of ifttt: Ecosystem, usage, and performance. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 398–404, New York, NY, USA, 2017. Association for Computing Machinery.

[47] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the Internet of Things (IoT), 2015. IEEE Internet Initiative.

[48] Shlomi Oberman Moshe Kol. Ripple20 CVE-2020-11896 RCE CVE-2020-11898 Info Leak, June 2020.

[49] Jonathan Myers, Leonardo Babun, Edward Yao, Sarah Helble, and Patrick Allen. Madiot: Memory anomaly detection for the internet of things. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, New York, NY, USA, 2019. IEEE.

[50] Phil Neray. Azure Defender for IoT: Agentless Security for OT, September 2020. https://techcommunity.microsoft.com/t5/microsoft-defender-for-iot-blog/azure-defender-for-iot-agentless-security-for-ot/ba-p/1698679.

[51] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V. Krishnamurthy, Edward J. M. Colbert, and Patrick McDaniel. IotSan: Fortifying the Safety of IoT Systems. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 191–203, New York, NY, USA, 2018. Association for Computing Machinery.

[52] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad-Reza Sadeghi. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 756–767, New York, NY, USA, 2019. IEEE.

[53] Dan Noal. Support for secure elements in freertos, Oct 2019. https://aws.amazon.com/blogs/iot/support-for-secure-elements-in-freertos/.

[54] O. Novo. Scalable access management in iot using blockchain: A performance evaluation. *IEEE Internet of Things Journal*, 6(3):4694–4701, June 2019.

[55] Nikolaos V Sahinidis. BARON: A general purpose global optimization software package. *Journal of global optimization*, 8(2):201–205, 1996.

[56] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards blockchain-based auditable storage and sharing of IoT data. In *Proceedings of the 2017 on Cloud Computing Security Workshop*, CCSW '17, pages 45–50, New York, NY, USA, 2017. ACM.

[57] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62, New York, NY, USA, 2013. IEEE.

[58] Eric Tucker. US seizes 2 domain names used in cyberespionage campaign, June 2021. https://www.washingtonpost.com/politics/us-seizes-2-domain-names-used-in-cyberespionage-campaign/2021/06/01/9c72cb2c-c316-11eb-89a4-b7ae22aa193e_story.html.

[59] Lingyu Wang, Mengyuan Zhang, Sushil Jajodia, Anoop Singhal, and Massimiliano Albanese. Modeling network diversity for evaluating the robustness of networks against zero-day attacks. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 494–511, Cham, 2014. Springer International Publishing.

[60] Zhe Wu, Curtis Yu, and Harsha V. Madhyastha. Costlo: Cost-effective redundancy for lower latency variance on cloud storage services. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 543–557, Oakland, CA, May 2015. USENIX Association.

[61] Liang Xiao, Xiaoyue Wan, Xiaozhen Lu, Yanyong Zhang, and Di Wu. Iot security techniques based on machine learning: How do iot devices use ai to enhance security? *IEEE Signal Processing Magazine*, 35(5):41–49, 2018.

[62] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, HotNets-XIV, New York, NY, USA, 2015. Association for Computing Machinery.

[63] B. Yuan, C. Lin, H. Zhao, D. Zou, L. T. Yang, H. Jin, and C. Rong. Secure data transportation with software-defined networking and k-n secret sharing for high-confidence iot services. *IEEE Internet of Things Journal*, 7(9):7967–7981, 2020.

[64] M. Zhang, L. Wang, S. Jajodia, A. Singhal, and M. Albanese. Network diversity: A security metric for evaluating the resilience of networks against zero-day attacks. *IEEE Transactions on Information Forensics and Security*, 11(5):1071–1086, 2016.