SUPnP: Secure Access and Service Registration for UPnP-Enabled Internet of Things

Golam Kayas, Mahmud Hossain, Jamie Payton, *Member, IEEE*, and S. M. Riazul Islam

***Abstract*—The service-oriented nature of the Universal Plug-and-Play (UPnP) protocol supports the creation of flexible, open, and dynamic systems. As such, it is widely used in Internet-of-Things (IoT) deployments. However, the protocol's service access mechanism does not consider security from the first principles and is therefore vulnerable to various attacks. In this article, we present an in-depth analysis of the service advertisement, discovery, and access methods of the UPnP protocol stack and identify security issues in an IoT network. Our analysis shows that adversaries can perform resource exhaustion, buffer overflow, reflection, and amplification attacks by exploiting the vulnerabilities of the UPnP protocol. To address these issues, we propose a capability-based security model for UPnP to ensure secure discovery, advertisement, and access of the UPnP services that considers the resource limitations of IoT devices. Our analysis shows the effectiveness of the proposed model against potential attacks, and our experimental evaluation highlights the feasibility of implementing our Secure UPnP (SUPnP) protocol in a network of IoT devices, incurring minimal network and performance overhead.**

***Index Terms*—Access, discovery, Internet of Things (IoT), network attacks, registration, security, Universal Plug and Play (UPnP).**

I. Introduction

THE Internet of Things (IoT) is propelling a paradigm shift in next-generation computing systems [1]. The IoT is rapidly becoming an essential element of applications across many domains, such as healthcare services, manufacturing industry, military domains, and transportation system, offering sensing, computation, and connectivity across a wide variety of smart devices [2]–[6]. The interest in IoT deployments continues to grow; the number of Internet-connected devices is projected to reach 24 billion by the end of 2020 [7].

Service-oriented architectures are well suited to support IoT-enabled systems, allowing IoT devices to advertise software (SW) services that leverage their unique sensing and actuation capabilities, which can then be discovered and used by applications. The Universal Plug-and-Play (UPnP) protocol has become widely used to support the dynamic advertisement, discovery, and access of SW services distributed across a network [8], [9]. UPnP is particularly well suited for IoT deployments that support opportunistic interactions across a large network of heterogeneous devices; the protocol offers interoperability, language independence, and decentralization in service-oriented pervasive networks [10] and supports the creation of open, scalable systems, requiring little to no configuration to deploy new IoT devices and to support discovery of the services that they offer.

However, a major limitation of the UPnP protocol is that it does not adequately address security, particularly with respect to service discovery, access control, and data integrity mechanisms. For example, an service device (SD) that provides services cannot verify the authenticity and integrity of a message send by a service consumer. Likewise, a control point (CP) that consumes services cannot verify the service provider. The detailed analysis of the UPnP security threats and vulnerabilities are provided in the next section. Research suggested that more than 20% of UPnP-enabled products are exposed to external and internal threats that take advantage of the UPnP protocol stack [11]–[14]. Network security scanners, such as Shodan and ZMap reported millions of vulnerable IoT devices around the globe where enabling UPnP is the root cause of the vulnerability [15], [16]. As such, enabling unsecured UPnP to support applications across IoT networks can have severe consequences, as illustrated by the Mirai, Qbot, and CallStranger attacks in recent years [17]–[19]. Therefore, securing the enormous number of IoT devices that use UPnP is a prime challenge for security researchers and application developers [12]. Compounding the challenge is the fact that most IoT devices are battery-powered and have limited computational capabilities, making it difficult to simply adapt existing security solutions [20]–[22] and to build secure models for UPnP-enabled IoT (UIoT) devices.

In this article, we analyze the security vulnerabilities of UPnP service discovery, advertisement, eventing, and control methods in IoT networks. The current implementation of UPnP cannot verify the capability of the service provider to provide an advertised service or identify if a consumer is authorized to use a service. We, therefore, identify various network attacks that adversaries can launch exploiting these vulnerabilities. We propose a capability-based security method to protect the UIoT devices from adversarial activities

Manuscript received December 24, 2020; accepted January 27, 2021. Date of publication February 11, 2021; date of current version July 7, 2021. This work was supported in part by the U.S. National Science Foundation under Grant CNS-1828363, and in part by the Sejong University Research Faculty Program under Grant 20212023. *(Golam Kayas, Mahmud Hossain, Jamie Payton, and S. M. Riazul Islam contributed equally to this work.) (Corresponding author: Golam Kayas.)*

Golam Kayas and Jamie Payton are with the Department of Computer and Information Science, Temple University, Philadelphia, PA 19122 USA (e-mail: golamkayas@temple.edu; payton@temple.edu).

Mahmud Hossain is with the Department of Cybersecurity Engineering, Visa Inc., Foster City, CA 94404 USA (email: mahhossa@visa.com).

S. M. Riazul Islam is with the Department of Computer Science and Engineering, Sejong University, Seoul 05006, South Korea (e-mail: riaz@sejong.ac.kr).

Digital Object Identifier 10.1109/JIOT.2021.3058699

2327-4662 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

that exploit service advertisement, discovery, eventing, and control vulnerabilities of UPnP that is feasible for implementation in IoT networks. Our proposed scheme, Secure UPnP (SUPnP), assigns a capability token to a UIoT device, which acts as an credential of that device. The capability token is cryptographically secure and cannot be forged. A UIoT device needs to present the token to a peer device in the IoT network for exchanging UPnP messages. The proposed model allows a service consumer device to verify the service provider device's capability to serve certain services. Similarly, it also enables the service provider device to verify the authorization of a service consumer device to consume requested services. Furthermore, a service provider device can verify the authenticity and integrity of a particular operation requested on a service. Thus, the proposed model serves to prevent the identified attacks in networks of UIoT devices. Finally, we also conduct an experimental evaluation using a prototype, emulating real-life scenarios to demonstrate the feasibility of deployment of this solution in IoT networks, exploring energy consumption, network throughput, request drop rate (RDR), and request response time (RRT).

Contribution: The contributions of this work are summarized as follows.

1) Despite the existence of some potential access-control solutions to deal with unauthorized access to UPnP services, the vulnerabilities associated with the service discovery methods in the context of UIoT are not yet addressed. We present a security analysis and identify vulnerabilities present in the stages of UPnP for service registration, advertisement, discovery, eventing, and control in UIoT systems.
2) To mitigate vulnerabilities in UIoT systems, we propose SUPnP, a capability-based security scheme for UPnP. We present algorithms for device enrollment, service registration, and capability verification as part of the SUPnP protocol.
3) We show that the proposed SUPnP scheme protects UIoT devices against network attacks with respect to trustworthiness, impersonation, authentication, and message freshness.
4) We implement a prototype of the SUPnP protocol and conduct an experimental evaluation. Our experiments, which focus on energy consumption, throughput, message RDR, and response time and demonstrate the feasibility of adopting SUPnP in UIoT settings.

Organization: The remainder of this article is organized as follows. Section II presents background on the UPnP vulnerabilities and the attacks exploit these vulnerabilities. Section III describes the proposed SUPnP scheme. The security analysis of the proposed SUPnP model is given in Section IV, followed by the experimental evaluation in Section V. Section VI presents related works with a comparative discussion and draw conclusions in Section VII.

II. Threat Model

The design of the UPnP protocol does not consider security with respect to service discovery, advertisements, actions, and events among IoT devices. Although several access control methods [23]–[28] have been proposed to prevent unauthorized access to IoT services, the vulnerabilities of the service discovery and access methods in UPnP have been largely overlooked. In this section, we derive a threat model for its use; a quick reference to the notations is provided in Table I for convenience. We begin by presenting a brief overview of the UPnP model.

A UPnP network includes two primary types of devices: 1) SD and 2) CP. Each SD offers one or more services for use in the UPnP network. CPs act as users of the services provided by the SDs. Fig. 1 shows the phases of UPnP protocol. In the first phase, SDs perform the *advertisement* of services while CPs perform *discovery* of advertised services. The UPnP standards do not provide any suitable mechanism

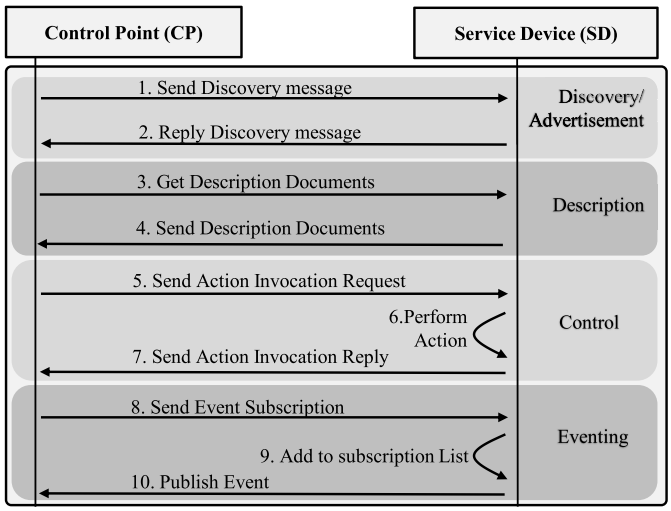

Fig. 1. Phases of UPnP.

TABLE I
Notations Used in This Work

Notation	Description
SD	Service-device
CP	Control-point
RA	Registration Authority
UCA	UPnP Certification Authority
DSD	Device Specification Document
SAD	Service Action Document
URL	Uniform Resource Locator
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
HTTP	Hypertext Transfer Protocol
UIoT	UPnP enabled IoT devices
URI	Uniform Resource Identifier
CapToken	Capability Token assigned to a participant of the network
PK_{RA}, SK_{RA}	Public and Secret Key of the Registration Authority
PK_{SD}, SK_{SD}	Public and Secret Key of a Service-device
PK_{CP}, SK_{CP}	Public and Secret Key of a Control-point
SK_P	$SK_P \in \{SK_{SD}, SK_{CP}\}$
$Cert_{UCA}$	Certificate of the UPnP Certification Authority
$Cert_{SD}$	Certificate of a Service-Device
$Cert_{CP}$	Certificate of a Control-Point
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
JVM	Java Virtual Machine
CPU	Central processing unit
RPL	Routing Protocol for Low-Power and Lossy Networks
DoS	Denial of Service

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

Advertisement	Discovery	Event Subscription	Control Message
NOTIFY * HTTP/1.1 HOST: 239.255.255.250:1900 CACHE-CONTROL: *expiration time* LOCATION: */descriptiop.xml* NT: *service:serviceType* USN: identifier for the *advertisement*	M-SEARCH * HTTP/1.1 HOST: 239.255.255.250:1900 MAN: ssdp:discover MX: *seconds to delay response* ST: *service:serviceType*	SUBSCRIBE publisher path HTTP/1.1 HOST: *hostname:portNumber* CALLBACK: *delivery URL* NT: upnp:event TIMEOUT: *Second-requested subscription* STATEVAR: *CSV of Statevariables*	POST path control URL HTTP/1.1 HOST: *hostname:portNumber* CONTENT-LENGTH: *bytes in body* CONTENT-TYPE: text/xml; charset="utf-8" SOAPACTION: service#actionName

Fig. 2. UPnP messages.

TABLE II
THREAT MODEL

Vulnerability	Adversary	Target	Scenario	Impact
Lack of service capability verification	Service Device	Control Point	Send faked or modified advertisement messages, advertising the fake services which is not really available	• False data injection • Redirection to malicious URL • Service impersonation
Lack of control capability verification	Control Point	Service Device	Anyone can send discovery message, no authentication or authorization is required. So the service device can not verify the control point capabilities.	• Sensitive data leakage • Denial of service • Control Point impersonation
No discovery verification	Control Point	Service Device	Send frequent Discovery to the rest of network devices	• Power Exhaustion • Denial of service • CPU Exhaustion
No advertisement authentication	Service Device	Control Point	Send frequent unnecessary advertisement messages	• Power Exhaustion • Denial of service • CPU Exhaustion
No integrity check in description	Control point	Service device	Spoofed description request is send, where the forged source is the target. In reply, the service description is send to the target which has to receive the malicious device description.	• Reflection & Amplification of the malicious traffic. • Distributed Denial of Service
Lack of integrity check in eventing	Control Point	Service Device	Frequent unnecessary event subscription is requested to overflow the target device.	• Denial of Service • Memory Overflow • Control-flow Hijack

for the verification of the multicast service discovery messages or the authorization of the sender. Next, in the *description* phase, a CP uses the information received in an advertisement message to send a description request in order to collect detailed information about an SD and the services it provides. The description documents sent in response include the name of the actions supported by the service, the parameters of the actions, the states of the service that can be monitored, etc. The UPnP protocol does not include any measures to verify the description documents received by the CP. After receiving all the required information about the SD and the services it provides, the CP can invokes a particular action on the SD in the *control* phase. Alternatively, the CP can subscribe to monitor a specific state of the service by invoking the *eventing* phase. Both *control* and *event subscription* messages are vulnerable to unauthorized requests, as the SD cannot verify and authorize these requests.

A malicious SD can exploit vulnerabilities in the UPnP to provide forged services. The vulnerabilities can also be exploited by a malicious CP to trick an SD and perform adversarial activity. In this section, we provide the details of the vulnerabilities and the attacks that malicious SDs and CPs can perform by exploiting the vulnerabilities of the UPnP discovery, description, eventing, and control phases in IoT networks. Table II presents a summary of the vulnerabilities and attacks.

A. Vulnerable Advertisement and Description

In UPnP, any participant can start advertising a service in the network. Moreover, an interested CP cannot verify an advertisement message and the sender SD of that advertisement message. Exploiting this vulnerability, an malicious participant can start advertising false service performing service impersonation. Moreover, a malicious advertisement can provide a malicious description location and fool the CP by redirecting to a malicious URL. These forgeries can lead to different service impersonation attacks. For example: a compromised refrigerator can impersonate a security camera of a smart home to provide a false service: `MonitorOccupancy`. When the home owner searches for a service to monitor the home, the compromised refrigerator advertises the `MonitorOccupancy` service. When a service offered by the compromised refrigerator is discovered and used, the compromised refrigerator device can provide arbitrary readings to the home owner in the reply of action invocations; for example, the impersonation of the `getLastOccupancy` service could be used to mask a potential physical security breach in the home.

1) Service Impersonation Using Device Description Forgery: An SD announces its services periodically to the known multicast address `239.255.255.250:1900` through a service advertisement message (Fig. 2). A CP

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

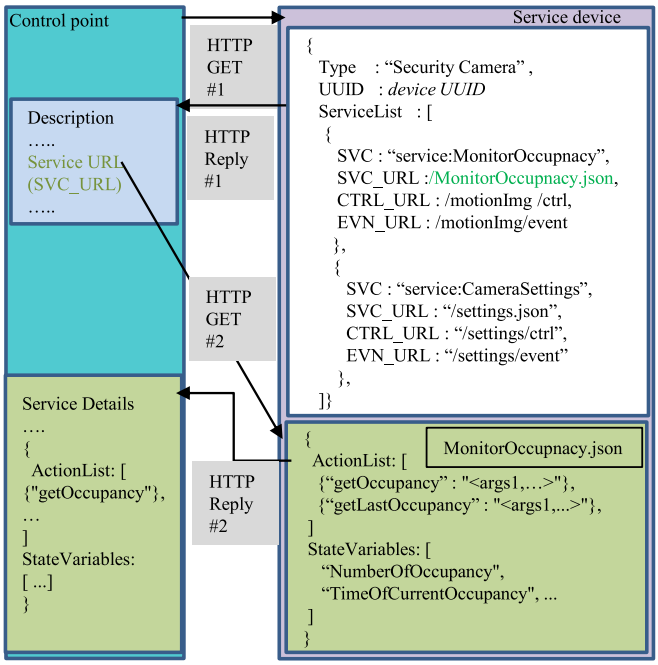

Fig. 3. UPnP description architecture.

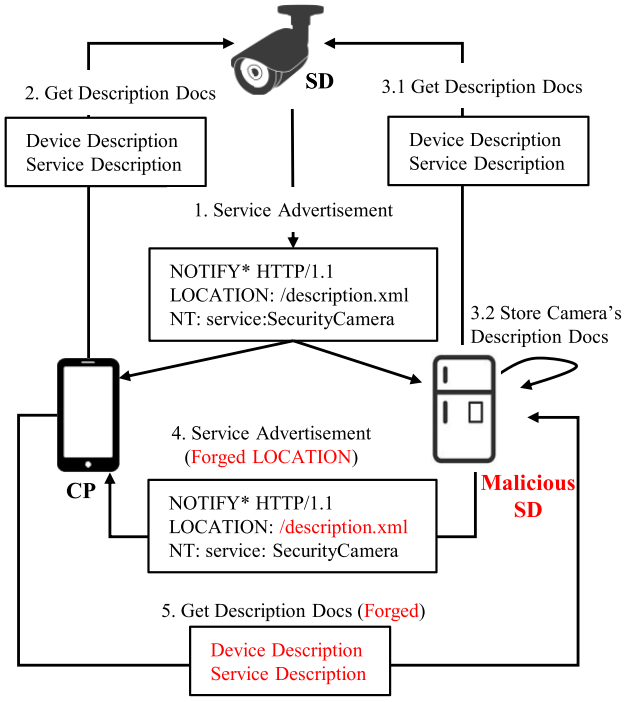

Fig. 4. Service impersonation using LOCATION forgery.

listens to this address to receive advertisement messages. Upon receipt of an advertisement message, the CP retrieves the sender's device description document (DDD) using the URI specified in the `LOCATION` field of the advertisement. The UPnP description architecture is shown in Fig. 3. The CP first sends an `HTTP GET` request to retrieve the DDD, which includes the description of the device and a list of services. Then, the CP sends another `HTTP GET` request to get the service description of the interested service. The location of the service description document (SDD) is mentioned in the `SVC_URL` field of the DDD.

A malicious SD can modify its DDD to add additional services in the service list (Fig. 3) for which it does not have SW or hardware (HW) capabilities. In the current implementation of UPnP, the CP cannot verify the authenticity and integrity of the DDD and treats the malicious SD as a legitimate SD.

2) Service Impersonation Using Advertisement Forgery: A malicious SD can advertise a service without having the capability to provide the service. As shown in Fig. 4, a legitimate SD (security camera) advertises its services in the network. A CP and a malicious refrigerator receive the advertisement as the message is multicasted in the network. The malicious refrigerator device retrieves the security camera's device description and SDDs using a `LOCATION URI` (e.g., `GET http://securitycamera-ip:port:/description.xml`) and stores it locally. A URI represents an absolute location of a resource. For instance, the `LOCATION` field of an advertisement message specifies the relative location of a resource, such as `/description.xml` or `/service.xml`. A CP constructs the `LOCATION` URI for the resource by prepending the IP address and port information to the `LOCATION`: `http://ip:port/resource.xml`. Next, the refrigerator replays the advertisement message to impersonate the security camera. The CP receives the replayed advertisement message and believes that the refrigerator provides the security camera's services, as the CP cannot verify the authenticity of the `LOCATION` URI for the advertisement message. The CP retrieves the device and SDD using the `LOCATION` URI of the replayed advertisement: `GET http:/refrigerator-ip:port:/description.xml`. At this point, the CP may attempt to invoke service requests directed to the impersonating device.

An adversarial CP can impersonate a legitimate CP to prevent an SD from carrying out important tasks. For example, a smartphone can impersonate another control device associated with the owner of a smart home and can send repeated requests to security camera of the house, which provides a very important service: `MonitorOccupancy`. The adversary can prevent the security camera from providing the occupancy monitoring service by keeping it busy with answering adversarial service requests. Thus, a malicious CP can prevent a crucial SD from providing real-time data to a legitimate CP through impersonation of the CP. Moreover, the malicious CP can invoke actions to change the state of an SD. For example, an IoT-enabled pacemaker implanted in a patient's body can be affected by a malicious action invocation, such as `increaseHeartRate`, by an adversary impersonating a physician's CP, which can be life threatening for the patient.

B. Vulnerable Discovery

Any participant of the UPnP network can multicast a discovery message in the UPnP network. There is no authentication involved in sending and receiving the discovery message. For instance, an SD cannot verify the integrity of the discovery message it receives neither can authenticate the sender of

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

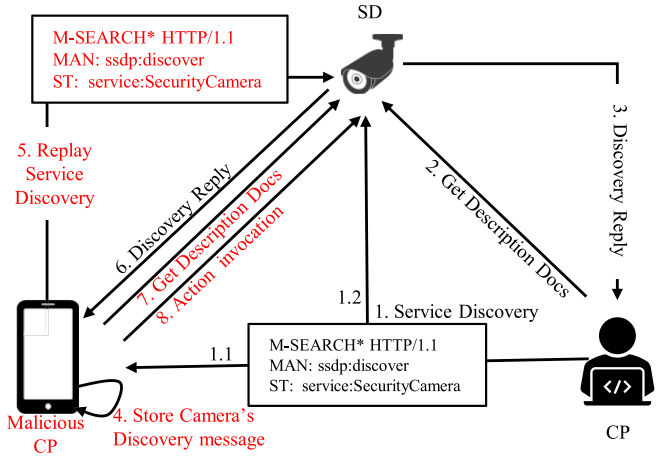

Fig. 5. CP impersonation attack.

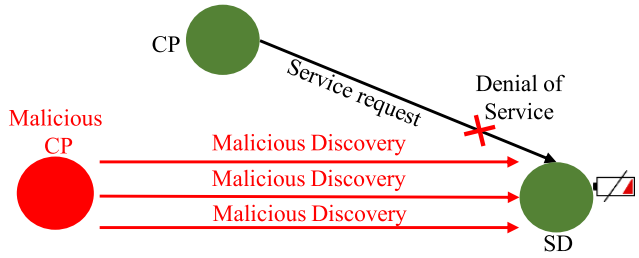

Fig. 6. DoS using discovery flooding.

the discovery message. An adversary takes advantage of this vulnerability and performs several malicious attacks.

1) Discovery Message Replay: In the current implementation of the UPnP, a malicious CP can interact with an SD, although it does not have capabilities to process the data provided by the service. As shown in Fig. 5, a legitimate CP broadcasts a discovery request in the network looking for a security camera services. An SD and a malicious CP receive the discovery request. Later, the malicious CP replays the discovery message in the network searching for the security camera services, impersonating the real CP. The SD replies to the malicious discovery request as it cannot validate the CP's authorization to use its services. The CP uses the information included in the discovery response to perform various actions on the SD, although it does not process the action responses as expected.

2) Search Target Forgery: Suppose that a CP can interact with a given IoT device, for example, a thermostat. Now, suppose that the CP is malicious and wants to search for security camera services, even though it cannot process the data returned from the services. The CP forges a discovery message by setting the ST field to `"service:SecurityCamera"` and multicasts it in the network. A security camera receives the discovery request and replies, as it cannot detect the forged request. Thus, a malicious CP tricks the SD into thinking that the discovery message is sent by a legitimate CP.

3) Discovery Flooding by Control Point: A malicious CP sends a large number of discovery message in a short span of time to search for a service in the network (see Fig. 6). An SD receives the malicious discovery messages and replies to requests. Receiving a huge number of discovery requests and

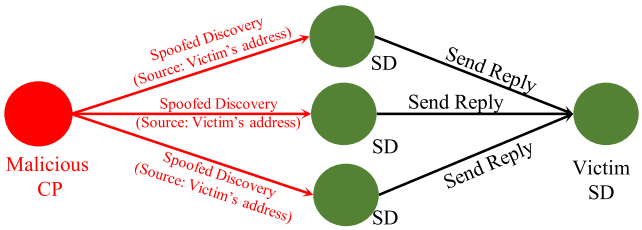

Fig. 7. Reflection and amplification using malicious discovery requests.

replying to them can exhaust the CPU and radio transceiver of the SD and prevent the SD from processing legitimate requests. Moreover, the processing of the malicious requests increases the power consumption of the SD, which can cause a significant decrease of the battery life of the SD. However, the current implementation of the UPnP does not have methods to defend against such flooding attacks to protect the IoT SDs.

4) Reflection and Amplification Using Spoofed IPs: The UPnP device and service discovery request is vulnerable to reflection and amplification attack in the discovery phase. As shown in Fig. 7, a malicious CP sends a discovery request to UPnP network searching for a service. The CP uses HTTPU (HTTP over UDP) to make the request. Unlike TCP, UDP packets are vulnerable to source address spoofing. Leveraging this fact, the malicious CP uses a victim device's IP address as the source of the UDP request. Consequently, an SD sends the discovery reply to the victim device. A study shows this spoofed discovery request amplify the reflected traffic from an SD to target up to 30.8 times [29]. Moreover, the service discovery message is muliticasted to a standard address and port (`239.255.255.250:1900`) in the network according to the UPnP standards [30]. So, all the SD providing the service in the network will receive the discovery request and reply to the victim device. This incredibly large amount of malicious traffic is used to generate a DDoS attack by reflecting the amplified traffic to the target's address.

C. Vulnerable Eventing

A CP subscribes to get notification of a state change event of a service by sending a event subscription request (see Fig. 2). The SD that accepts the event subscription request will publish the event to `CALLBACK URL` specified in the event subscription message. The SD cannot verify the sender of the event subscription message and its sender. Moreover, there is no mechanism to verify the `CALLBACK URL` used to publish the event as well. A attacker exploits these vulnerabilities and perform several adversarial activity.

D. Event Subscription List Overflow Attack

In the current implementation of the UPnP, an SD cannot verify the authenticity of an event subscription request. Once the service description is leaked an adversary has all the information (such as event URL) needed to send a event subscription message. Therefore, a malicious CP can subscribe to an event using a forged `CALLBACK` URL. As shown in Fig. 8, a malicious CP sends a large number of event subscription requests, each containing a

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

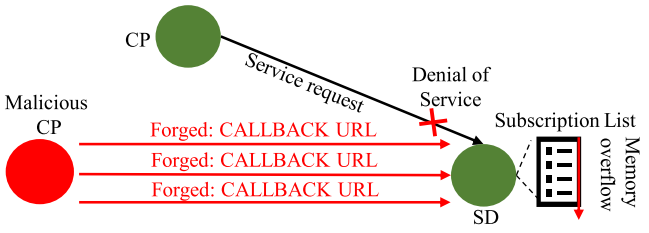

Fig. 8. Buffer overflow using forged event subscription requests.

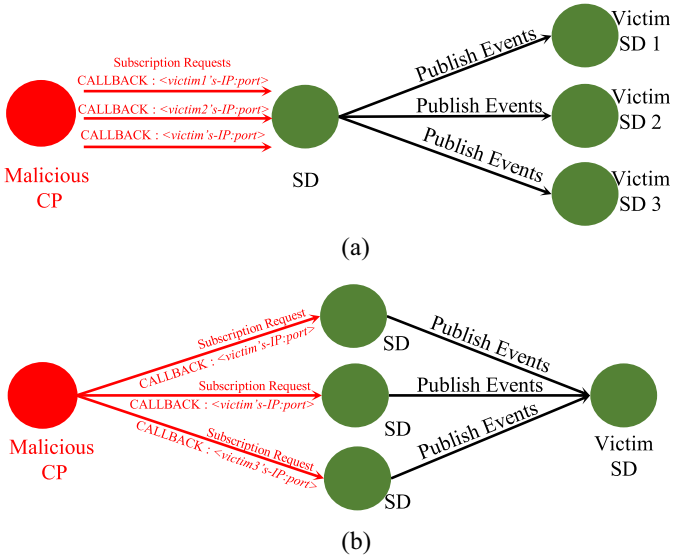

Fig. 9. (a) Reflection and (b) amplification using event subscription CALLBACK.

forged `CALLBACK` URL, to overflow the memory of an SD. An SD receives the request and stores it in a subscription list as a tuple `<subscription-UUID (SID), callback-URL, timeout, http-version>`. As the SD gets a huge number of malicious event subscription requests in a short period of time, the subscription list grows in such a way that it exhausts the SD's memory. As such, the SD cannot process any service requests.

E. Event URL Forgery Attack

A malicious CP can use a spoofed `CALLBACK` URL in the event subscription request. As the SD cannot verify the authenticity of the request, the SD accepts the request and publishes all the service events to the forged `CALLBACK` URL. As shown in Fig. 9 the malicious CP subscribes to events using spoofed `CALLBACK` URLs. The CP uses a spoofed `CALLBACK` URL to subscribe to events and enable SDs to publish their events to the victim devices.

III. Proposed Scheme

In light of the security threats identified, we propose a security scheme for UPnP that prevents malicious SDs from providing forged services and CPs from impersonating legitimate service consumers in IoT networks. Our proposed SUPnP protocol can identify forged discovery messages, advisement messages, DDDs, and SDDs. Our approach considers several access control models, including role-based access control (RBAC) [23], capability-based access control (CapBAC) [24], and attribute-based access control (ABAC) [26], adapting and integrating features as part of a model that is appropriate for use in IoT networks.

While each of these access control models offers benefits, when applied in isolation, they do not adequately address the needs for secure service advertisement, discovery, and usage in IoT networks. RBAC manages access based on a hierarchy of the rights assigned to the specific roles. This model allows multiple users to be grouped into roles that need access to resource, which can be useful for limiting the number of access policies but is not suitable for large, complex IoT networks. ABAC provides more fine-gained and contextualized access control, which is desirable for IoT scenarios. Access requests in ABAC are evaluated against a range of attributes that define the user, the action, the resource, and the context. Defining a set of generalized attributes can be difficult when applying this model in a heterogeneous and dynamic IoT network. Both RBAC and ABAC use a centralized approach. Though these models have been applied in IoT-specific scenarios, achieving end-to-end security using a centralized architecture on a distributed system such as the IoT can be quite challenging. Unlike other models, the CapBAC model is based on capability of the subjects. Every entity of an IoT system is given a unforgeable capability token that uniquely references an object as well as an associated set of access rights or privileges. A major advantage of this model is that distributed devices do not have to manage complex sets of policies or carry out elaborate authentication protocols, which makes it suitable for resource-constrained IoT devices.

Considering the nature of IoT networks, we choose a combination of CapBAC and ABAC models to achieve a fine-gained and distributed solution in our proposed model. SUPnP utilizes capability-based and ABAC methods to validate an SD's capability to provide a service and a CP's capability to interact with a service. Importantly, our approach is feasible for implementation in networks of resource-constrained IoT devices.

Fig. 10 presents an overview of the proposed scheme. In SUPnP, a trusted entity, such as the UPnP certification authority (UCA) [31], a device manufacturer, or a service provider, issues a device specification document (DSD) and a service action document (SAD) to an SD and a CP, respectively (step 1). A DSD contains an SD's capability to provide certain services. Likewise, a SAD includes a CPs capability to interact with a service. A DSD and SAD are signed and cryptographically protected; therefore, they cannot be forged. An SD has to contact a registration authority (RA) to provide services in a UPnP network (step 2). The RA validates the SD's DSD, DDD and SDD and ensures that the device can provide services that are specified in the SDD and DDD. The RA issues a capability token ($CapToken_{sd}$) to the SD. The device uses the $CapToken_{sd}$ as a proof of the service authenticity (step 4). Similarly, a CP submits its SAD to the RA for validation. The RA verifies the authenticity of the SAD and confirms that the CP can consume services that can be specified in its discovery messages.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

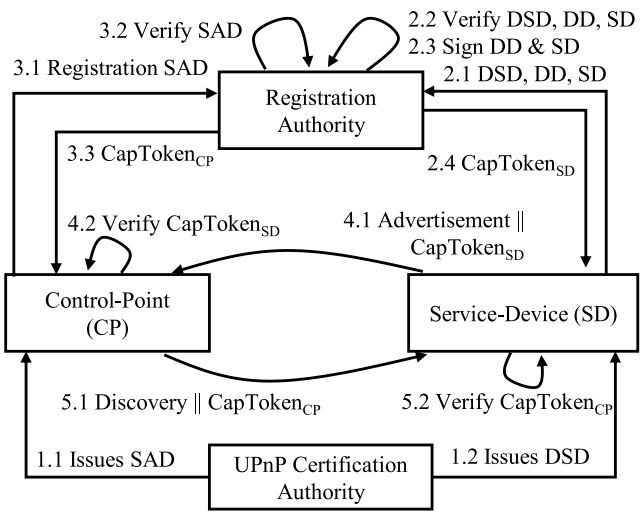

Fig. 10. Overview of SUPnP.

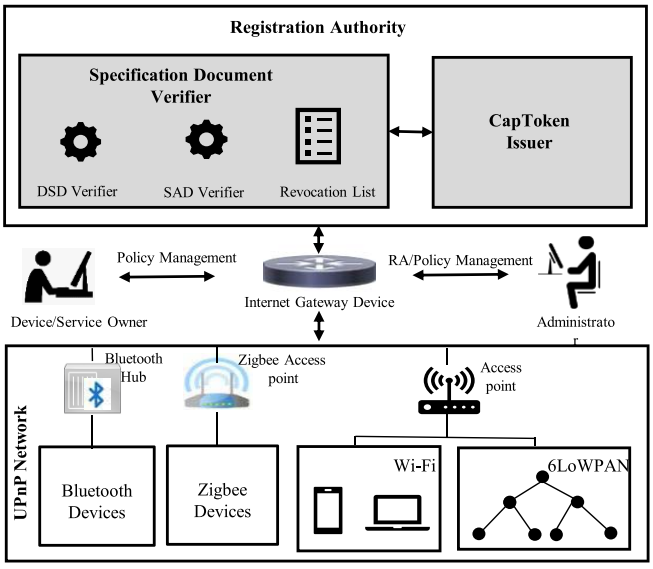

Fig. 11. Components of SUPnP.

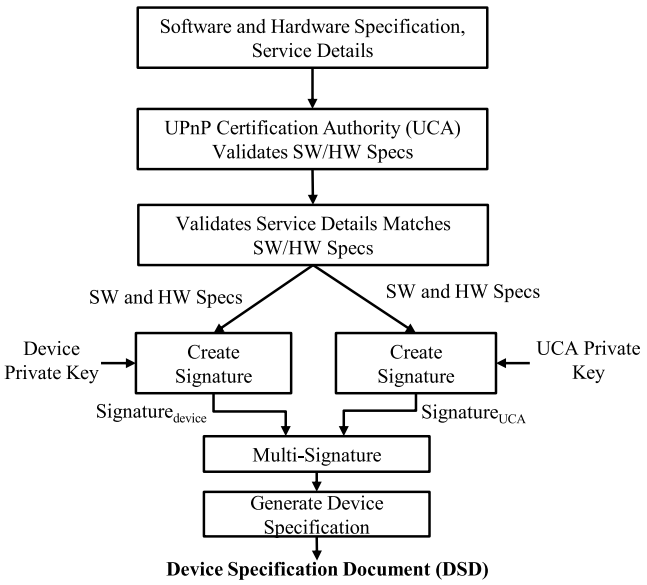

Fig. 12. DSD for SD.

```
"TYPE" : "SD",
"SD_PK": "048E63408...6844AD00"
"Name" : "SecurityCamera",
"HW"   :{"Name"   : "ReMote",
  "Model"  : 3,
  "CPU"    : "1.2 kHz(Single core)",
  "RAM"    : "128 kb",
  "STORAGE": "1 MB",
  "NET-INTERFACE": ["WiFi", "IEEE 802.15.4"]
}
"Software": {"OS": "Contiki", "Runtime": C},
"SERVICES" : [
 {"NAME-TYPE": "MonitorOccupnacy:monitor"},
 {"NAME-TYPE": "CameraSettings:settings"}],
"SIG-DEV" : "D4E5A9C27...F52059F",
"SIG-UCA" : "5A9C27D4E...A4C329A",
"SIG-VER-CON" : { "CON" : "2-of-2";
"SIGS": [SIG-OWNER, SIG-UCA]
}
```

Listing 1. Example DSD of an SD.

To this end, the RA issues $CapToken_{cp}$ to the CP which the device can use as the proof of its capability of consuming certain services (steps 3 and 5).

Fig. 11 presents the components of SUPnP. The SUPnP has three phases: 1) device enrollment; 2) service and device registration; and 3) service access. In the device enrollment phase, an SD and a CP are issued a DSD and a SAD, respectively. An SD's capability to provide services and a CP's authorization to use services are validated in the registration phase. Finally, the service access phase ensures the secure service discovery and advertisement. The details of the phases are provided as follows.

A. Device Enrollment

1) SD Enrollment: An SD needs to have a DSD provided by a trusted entity in order to provide a service in the UIoT network. To acquire the DSD for an SD, either the device manufacturer or the service provider can submit a request to the UCA. Fig. 12 presents the details on issuing a DSD to an SD. The UCA receives the HW and SW specifications of an SD from a manufacture or service provider. The UCA also receives the details of the services, such as service names, actions, and events, provided by the SD. The UCA validates the device specifications, ensuring that the device can provide the services specified in the service detail, and executes various test case scenarios to validate the operations of the services.

After the successful validation of the capabilities of the SD, the UCA generates a DSD for the SD and signs the DSD using the private key of its private (SK_{uca}) and public (PK_{uca}) key pair. Next, the requester issues a private (SK_{sd}) and public (PK_{sd}) key pair to the SD and signs the DSD using the SK_{sd}

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

of the device. Hence, the DSD is contains a 2-of-2 multisignature, requiring verification of both the signature of the UCA and the signature of the device to check the authenticity of the DSD document. The SD is also issued a certificate Cert_{sd} by a known certificate authority, such as DigiCert[1] or Verisign,[2] containing the public key PK_{sd} of the device. Additionally, the device is provided the certificate of the UCA Cert_{uca}. The Cert_{uca}, Cert_{sd}, SK_{sd}, PK_{sd}, and DSD document are stored in the device. These information are used to verify the authenticity of an SD's capability to provide certain services. Listing 1 shows an example DSD.

Components in the DSD include as follows.

1) *TYPE:* The type of the participant. Here, the type is "SD."
2) *SD_PK:* Public key of the SD.
3) *Hardware:* HW description of the device (e.g., CPU, RAM, ROM, and network interfaces).
4) *Software:* SW specification of the device (e.g., operating system and runtime environment).
5) *SERVICES:* The list of services, represented as `(name, type)` pairs, that are provided by the SD.
6) *SIG-OWNER:* The signature of owner, generated from the SAD document contents using the secret key of the SD.
7) *SIG-UCA:* The signature of the UCA, generated from the SAD document contents using the secret key of the UCA.
8) *SIG-VER-CON:* The verification condition of the `SAD`. The "`CON`" field value "`2-of-2`" means both signatures mentioned in the "`SIGS`" field need to be verified to prove the authenticity of this document.
9) *SIGS:* The signatures need to be verified to check the authenticity of this document.

2) CP Enrollment: A CP device runs an application that interactions with an SD. The UCA is contacted with the list of the services the CP wants to access along with the actions it can perform and a list of events it can subscribe. The UCA validates the capabilities of the CP to interact with the provided services and issues a SAD document. The UCA signs the SAD document using its private key SK_{uca}. The application developer or provider of the CP also signs the SAD document using the private key of the CP's private (SK_{cp}) public (PK_{cp}) key pair. Hence, the SAD is signed with a 2-of-2 multisignature. The CP is also issued a certificate (Cert_{cp}), which is signed by a known certificate authority, containing PK_{cp} so that the CP can use it to proof the authenticity of its key pair. An example of the SAD is shown in Listing 2. The fields of the SAD are as follows.

1) *TYPE:* The type of the participant. Here, the type is "CP."
2) *Name:* A user-friendly name of the participant given by the manufacturer.
3) *CP_PK:* The public key of the CP.
4) *SERVICES:* The list of services, represented as `(name, type)` pairs, that the CP will be authorized to use.

[1]https://www.digicert.com/
[2]https://www.verisign.com/

```
"TYPE" : "CP",
"Name" :  "Smart Phone",
"CP_PK"  : "048E63408...6844AD00"
"SERVICES": [
 {"NAME-TYPE": "MonitorOccupnacy:monitor"},
 {"NAME-TYPE": "CameraSettings:settings"}
],
"SIG-CP"  : "D4E5A9C27...F52059F",
"SIG-UCA" : "5A9C27D4E...A4C329A",
"SIG-VER-CON" : { "CON" : "2-of-2";
"SIGS": [SIG-OWNER, SIG-UCA]
}
```

Listing 2. Example SAD of a CP.

RA Discovery	RA Discovery Response
M-SEARCH * HTTP/1.1 HOST: 239.255.255.250:1900 MAN: ssdp:discover MX: *seconds to delay response* ST: RA	NOTIFY * HTTP/1.1 HOST: 239.255.255.250:1900 Time to Live: *12 Hour* LOCATION: *<RA-IP>:port* NT: RA NTS: ssdp:alive USN: identifier for the advertisement

Fig. 13. RA discovery messages.

5) *SIG-OWNER:* The signature of the owner is generated from the content of the SAD document using the secret key of the CP.
6) *SIG-UCA:* The signature of the UCA, generated from the SAD document contents using the secret key of the UCA.
7) *SIG-VER-CON:* The verification condition of the `SAD`. Here, the "`CON`" field value "`2-of-2`" means both signatures mentioned in the "`SIGS`" field need to be verified to prove the authenticity of this document.
8) *SIGS:* The signatures need to be verified to check the authenticity of this document.

B. Registration Process

1) Registration Authority Discovery: The RA is a UPnP service, which is hosted by the Internet Gateway device [32] or a standalone device, such as a workstation or a server, on the network. When a new participant (SD or a CP) enters a UPnP network, it needs to discover the RA service. The participant sends an `SSDP M-SEARCH` message to the known multicast address `239.255.255.250` on port 1900 via the UDP protocol (see Fig. 13).

The RA replies with a unicast response that contains the location (`IP:Port`) of the RA service (Fig. 13). After the RA discovery, the participant sends its specification document (DSD or SAD) to the RA, which can validate the authenticity of the document and the authorization of the participant to provide or use services. After the successful verification of the document and capability, the participant is issued a `CapToken` so that it can provide or access services. Fig. 14 gives an overview of the registration process.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

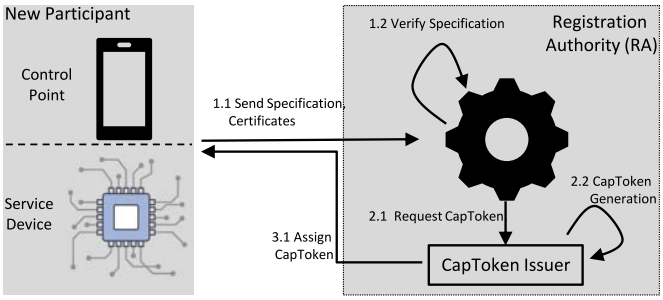

Fig. 14. Registration process.

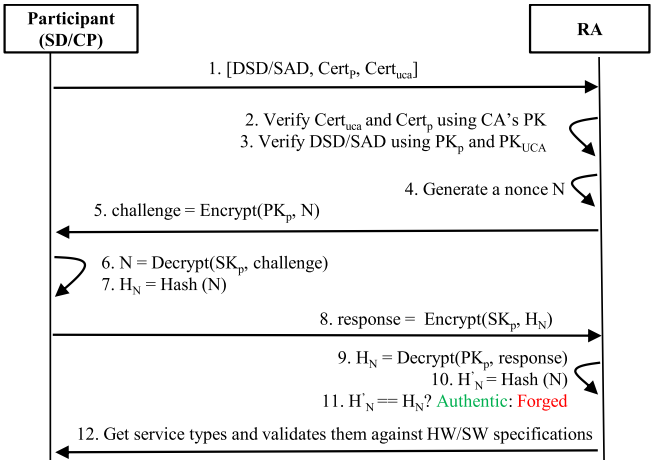

Fig. 15. DSD/SAD verification process.

2) Verification of Specification Document (DSD/SAD): Fig. 15 presents the messages exchanged between a new participant (SD or a CP) the RA for the verification of its specification document. The details of the verification steps shown in Fig. 15 are as follows.

1) *Step 1:* A participant sends its DSD or SAD, Cert_p, and Cert_{uca} to the RA, such that $\text{Cert}_p \in \{\text{Cert}_{sd}, \text{Cert}_{cp}\}$. The Cert_{sd} and Cert_{cp} are the certificates of an SD and a CP, respectively.
2) *Step 2:* The RA validates the authenticity of the participant's public key PK_p and the UCA's public key PK_{uca} included in the Cert_p and Cert_{uca}, respectively, by verifying the signatures of these certificates. Note that $PK_p \in \{PK_{sd}, PK_{cp}\}$, and PK_{sd} and PK_{cp} are the public keys of an SD and CP, respectively.
3) *Step 3:* The RA verifies the authenticity and integrity of the specification document DSD or SAD. The RA validates the multisignature "SIG-DEV," and "SIG-UCA" using the public key PK_d and PK_{uca}, respectively, as a DSD or SAD is a 2-of-2 multiple signature document.
4) *Steps 4 and 5:* The RA needs to verify that a DSD (or SAD) document is really belong an SD (or a CP). To make sure that the participant is legitimate, the RA sends a challenge to the participant. The RA generates a nonce N and encrypts the challenge using the public key of the participant as $\texttt{challenge} = \texttt{Encrypt}(PK_p, N)$. The RA sends that challenge to the participant.

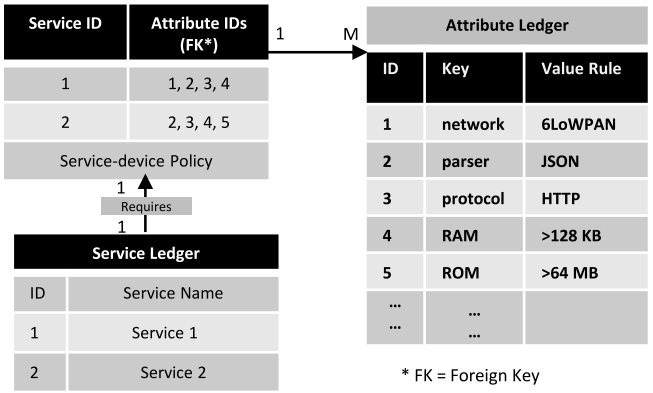

Fig. 16. Example of an attribute ledger.

5) *Steps 6–8:* The participant decrypts the challenge using its secret key SK_p to retrieve the nonce N. Note that $SK_p \in \{SK_{sd}, SK_{cp}\}$, and SK_{sd} and SK_{cp} are the private keys of an SD and CP, respectively. Next, the participant generates a signed response to challenge by encrypting the hash of the nonce N ($H_N = \text{Hash}(N)$) with the secret key of the device as $\texttt{response} = \texttt{Encrypt}(SK_p, H_N)$. The SD sends the response to the RA.
6) *Steps 9–11:* The RA decrypts the response using PK_p and retrieves H_N. The RA computes the hash of the nonce ($H'_N = \text{Hash}(N)$) and authenticates the participant if H' matches H. Otherwise, the RA determines that the DSD or SAD submitted by an SD or a CP is forged. In case of a verification failure in any step, the RA adds the SD or the CP to the revocation list.
7) *Step 12:* The RA needs to verify that the capability of an SD matches its DDD. The RA retrieves the device description document of the SD. The RA matches the services provided by the SD with its HW and SW specification included in the DSD. The RA uses an attribute ledger to perform the validation. The ledger maintains a mapping between a service type and the HW and SW attributes require to provide the service. Fig. 16 shows an example of the attribute ledger.

Through these steps, the RA authenticates the participant. The participant also can request the certificate of the RA.

3) CapToken Generation: The RA generates and assigns a capability token (CapToken) to a new participant after the success full verification of a specification document is passed. An SD presents its CapToken to a CP to show it is capable and authorized to provide the specified services. Similarly, a CP presents its CapToken to an SD to show that is authorized to use services provided by the SD.

CapToken Generation for SDs: The RA issues a CapToken to an SD. Listing 3 shows the fields of the CapToken. The RA uses Algorithm 1 to generate the CapToken. A number of signatures are included in the CapToken: the advertisement signature (ADVERTISEMENT-SIG), description signature (DESCRIPTION-SIG), and service signature (SERVICE-SIG).

The RA computes a DESCRIPTION-SIG as $\texttt{SIGN}(SK_{RA}, \texttt{device-description}) = \texttt{ENCRYPT}\ (SK_{RA}, \texttt{Hash}$

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

```
"ID" : "6E1S6F5F16RT51FSADF651"
"ISSUER_INSTANT" : "2017-09-15T12:10:47Z",
"RA_PK" : "048E...D314",
"SD_PK" : "0D98...E425",
"RA-SIG"  : "375F...F646",
"TYPE"  : "SERVICE-DEVICE",
"ADVERTISEMENT-SIG"  : "C27D...5C42",
"DESCRIPTION-SIG"    : "5C42...C27D",
"SERVICES" : [
 {"TYPE":"monitor","SERVICE-SIG":"D4...3D"},
 {"TYPE":"settings","SERVICE-SIG":"5C...F5"}
]
```

Listing 3. Example CapToken issued to an SD.

```
"ID"  : "6E1S6F5F16RT51FSADF651"
"ISSUER_INSTANT" :"2017-09-15T12:10:47",
"RA_PK"  : "048E...6844",
"CP_PK"  : "048E...D314",
"RA-SIG"  : "597A...D64E",
"TYPE"  : "CONTROL-POINT",
"CAPTOKEN-LOCATION-SIG" : "C27D...5C42",
"SERVICES" : [ "monitor", "settings" ]
```

Listing 4. Example CapToken for a CP.

(device-description-content)) to protect the integrity and authenticity of a DDD. The SK_{RA} and PK_{RA} are the RA's private and public keys, respectively. A DDD includes a list of services an SD intends to provide. The modification in the service list will invalidate the DESCRIPTION-SIG. Hence, the verification of the DESCRIPTION-SIG enables a CP to identify forged services.

Next, the RA computes a SERVICE-SIG as SIGN(SK_{RA}, service-description) = SIGN(SK_{RA}, Hash (service-description-content)) to protect the integrity and authenticity of an SDD. As shown in Listing 3, the RA computes a SERVICE-SIG for each of the services provided by an SD. An SDD contains a list of actions that are used to control (or interaction with) an SD. The modification in the SDD (add or remove actions) will invalidate the SERVICE-SIG signature. Hence, the verification of the SERVICE-SIG enables a CP to identify forged actions.

As shown in Listing 3, the RA also computes an ADVERTISEMENT-SIG, which protects the integrity of the locations URIs of the device description and the CapToken. The RA computes the ADVERTISEMENT-SIG as SIGN(SK_{RA}, device-description-uri, captoken-uri) = ENCRYPT(SK_{RA}, Hash (device-description-uri || captoken-uri)). A CP uses captoken-uri to retrieve a CapToken issued to the SD and the device-description-uri to fetch the DDD that includes the list of services that the SD provides. The captoken-uri and device-description-uri are the absolute locations of the CapToken and device-description document, respectively, and are expressed as

Algorithm 1 CapToken Generation for SD

```
1: procedure SD_TOKEN_GENERATION(sdInfo)
2:     /* Get RA public PK_RA, private key SK_RA */
3:     (PK_RA, SK_RA) = getKeyPair();
4:     ip-port = append(sdInfo.IP, sdInfo.port);
5:     capToken = getCapTokenInstance();
6:     capToken.ID = getRandomID();//Generate token ID
7:     capToken.RA_PK = PK_RA; //Add RA Public key
8:     /*Retrieve the device specification info*/
9:     specInfo = sdInfo.getDSD();
10:    /*Generate and add SERVICE-SIGs to CapToken */
11:    serviceList = specInfo.getServiceList();
12:    for each svc in serviceList do
13:        svcSig = Sign(SK_RA, Hash(svc.description));
14:        capToken.addService(svcSig, svc.type);
15:    end for
16:    /*Compute and add DESCRIPTION-SIG.*/
17:    des-uri = append(ip-port, sd.askDescriptionURI());
18:    capToken.DESCRIPTION-SIG = Sign(SK_RA,
19:        Hash(des-uri.content));
20:    /*Compute ADVERTISEMENT-SIG.*/
21:    token-uri= append(ip-port, getCapTokenLocation());
22:    capToken.ADVERTISEMENT-SIG=Sign(SK_RA,
23:        Hash(description-uri || token-uri));
24:    /*Generate and add RA-SIG to CapToken */
25:    capToken.RA-SIG=Sign(SK_RA, Hash(capToken.content));
26:    return capToken;
27: end procedure
```

Algorithm 2 CapToken Generation for CP

```
1: procedure CP_TOKEN_GENERATION(cpInfo)
2:     /* Get RA public (PK_RA), private key (SK_RA)*/
3:     (PK_RA, SK_RA) = getKeyPair();
4:     ip-port = append(cpInfo.IP, cpInfo.port);
5:     capToken = getCapTokenInstance();
6:     capToken.ID = getRandomID();//Generate token ID
7:     capToken.RA_PK= PK_RA; //Add RA Public key
8:     /*Retrieve the control point specification info*/
9:     specInfo = cpInfo.getSAD();
10:    /*Add service types to CapToken*/
11:    serviceList = specInfo.getServiceList();
12:    for each svc in serviceList do
13:        capToken.add(svc.type);
14:    end for
15:    /*Generate and add CAPTOKEN-URI-SIG */
16:    token-uri= append(ip-port, getCapTokenLocation());
17:    capToken.CAPTOKEN-URI-SIG =
18:        Sign(SK_RA,Hash(captoken-uri));
19:    /*Generate and add RA-SIG to CapToken */
20:    capToken.RA-SIG=Sign(SK_RA, Hash(capToken.content));
21:    return capToken;
22: end procedure
```

captoken-uri = http://sd-ip:port/captoken.json and device-description-uri = http://sd-ip:port/description.xlm. To this end, the RA signs the CapToken (RA-SIG) to protect the integrity and authenticity of the CapToken's contents.

CapToken Generation for CPs: Listing 4 shows the details of the fields of a CapToken issued to a CP. The RA uses Algorithm 2 to issue a CapToken to a CP. The RA issues a SERVICE-SIG for each of the service the CP can consume. The RA computes the SERVICE-SIG as SIGN(SK_{RA}, service-name, service-type) = ENCRYPT(SK_{RA},

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

Hash (service-name || service-type)). The RA uses the service names and types included in a CP's SAD document to compute the SERVICE-SIG (see Listing 2). Next, the RA computes a CAPTOKEN-LOCATION-SIG as SIGN(SK_{RA}, captoken-uri) = ENCRYPT(SK_{RA}, Hash (captoken-uri)). A CP serves its CapToken from the captoken-uri, which is expressed as http://cp-ip:port/captoken.json. The CP attaches the CAPTOKEN-URI-SIG with a discovery, action, and event subscription requests. Hence, an SD is protected from processing forged discovery, action, and event requests sent from a malicious device with spoofed IP addresses. After the generation of all necessary signatures, the RA prepares a CapToken document and signs it to protects the integrity and authenticity of the CapToken.

C. Capability Enforcement and Secure Communication

After the participant is finished the registration process successfully a CapToken assigned by the *RA*. The CapToken, is used in the verification of the messages send by the participant. In this section, we will explain the capability enforcement process in UPnP operations.

1) Secure Advertisement: An SD periodically sends advertise messages in the network to announce its services. Fig. 17 shows the advertisement message sent by an SD. The SD adds two additional fields in the advertisement message: 1) CapToken location (CAPTOKEN-LOCATION) and 2) advertisement signature (ADVERTISEMENT-SIG). Upon receiving an advertisement, a CP validates the signature using the public key (PK_{RA}) of the *RA* as Hash(CAPTOKEN-LOCATION URI || LOCATION URI) == DECRYPT(PK_{RA}, ADVERTISEMENT-SIG) ? Authentic : Forged. The value of the LOCATION and CAPTOKEN-LOCATION fields specify the relative paths to the DDD and CapToken, respectively. The CP constructs the LOCATION URI as http://sd-ip:port/LOCATION and the CAPTOKEN-LOCATION URI as http://sd-ip:port/CAPTOKEN-LOCATION. If the ADVERTISEMENT-SIG verification results in Authentic, then the CP ensures that the location of the CapToken (CAPTOKEN-LOCATION) or the location of the device description (LOCATION) is not forged. Hence, the CP validates the authenticity and integrity of the advertisement message.

2) Secure Description: After the successful verification of ADVERTISEMENT-SIG, the CP retrieves the CapToken from the CAPTOKEN-LOCATION. The CP validates RA-SIG using the RA's public key PK_{RA}. Hence, the CP ensures the authenticity and integrity of the CapToken. Next, the CP retrieves the DDD using the location URI (LOCATION). The CP validates DESCRIPTION-SIG to ensure that the SD does not modify (added or removed services in the service list) the description document after the registration. The CP validates the DESCRIPTION-SIG as Hash(Device Description) == DECRYPT(PK_{RA}, DESCRIPTION-SIG) ? Authentic : Forged. The steps of the secure description process are shown is Fig. 17.

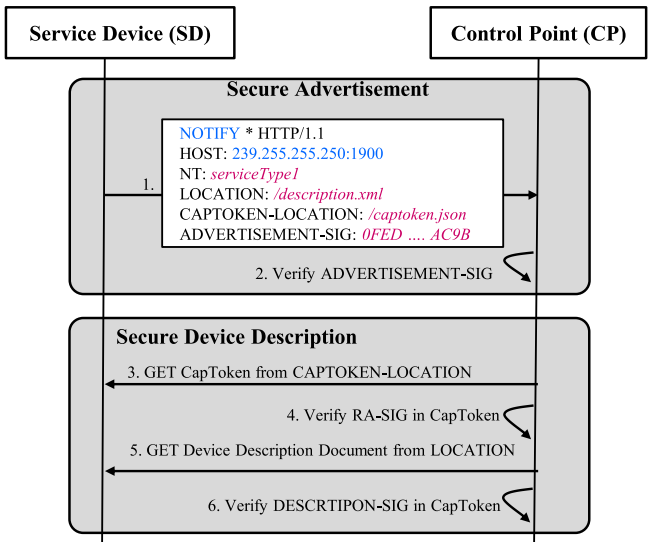

Fig. 17. Secure service advertisement.

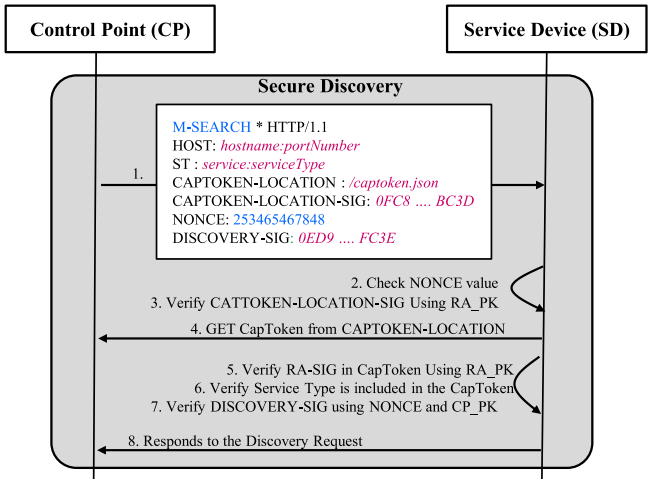

Fig. 18. Secure service discovery.

3) Secure Discovery: A CP sends a discovery request to find out a targeted service in the network. Fig. 18 shows the discovery message of SUPnP. The CP adds additional headers fields: CAPTOKEN-LOCATION, CAPTOKEN-LOCATION-SIG, DISCOVERY-SIG, and NONCE to an discovery request. Upon receiving a discovery message SD first checks the NONCE. If the NONCE value has seen before, the SD drops the discovery request. The value of the CAPTOKEN-LOCATION fields specify the relative paths to the CapToken. The SD constructs the CAPTOKEN-LOCATION URI as http://cp-ip:port/CAPTOKEN-LOCATION. The SD validates the CAPTOKEN-LOCATION-SIG using the public key of RA (PK_{RA}) as Hash(CAPTOKEN-LOCATION URI) == DECRYPT(PK_{RA}, CAPTOKEN-LOCATION-SIG) ? Authentic : Forged. If the CAPTOKEN-LOCATION-SIG verification results in Authentic, then the SD ensures that the location of the CapToken (CAPTOKEN-LOCATION) is not forged. After that, the SD retrieves the CapToken from the CAPTOKEN-LOCATION URI. The SD validates RA-SIG

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

TABLE III
PROPERTIES EVALUATED IN THE SECURITY ANALYSIS OF SUPNP

Security Property	Attack Scenario	Requirements
Trustworthy Capability Verification	An adversary sends a forge capability document(DSD, or SAD) during the registration process.	Registration Authority(RA) should be able to identify the forged capability document and reject registration request.
SD Impersonation mitigation	A malicious SD sends a forged advertisement with an altered service description document.	The control-point should detect the forgery of the advertisement and service description document.
CP Impersonation mitigation	A malicious CP sends a fake discovery request to find a service without having the capability to process the service data.	An SD should identify the fake discovery request and drop the request without processing it.
Action Authentication	An adversary gains unauthorised access to an SD's service description document, learns the control URL from the document, and sends a forged service action request.	The SD should be able to detect that the CP does not have the capability to perform the action.
Event Subscription Authentication	An adversary gains unauthorised access to an SD's device description document, learns the event URL from the document, and sends an event subscription request.	The SD should detect the unauthorized subscription request and ignore it.

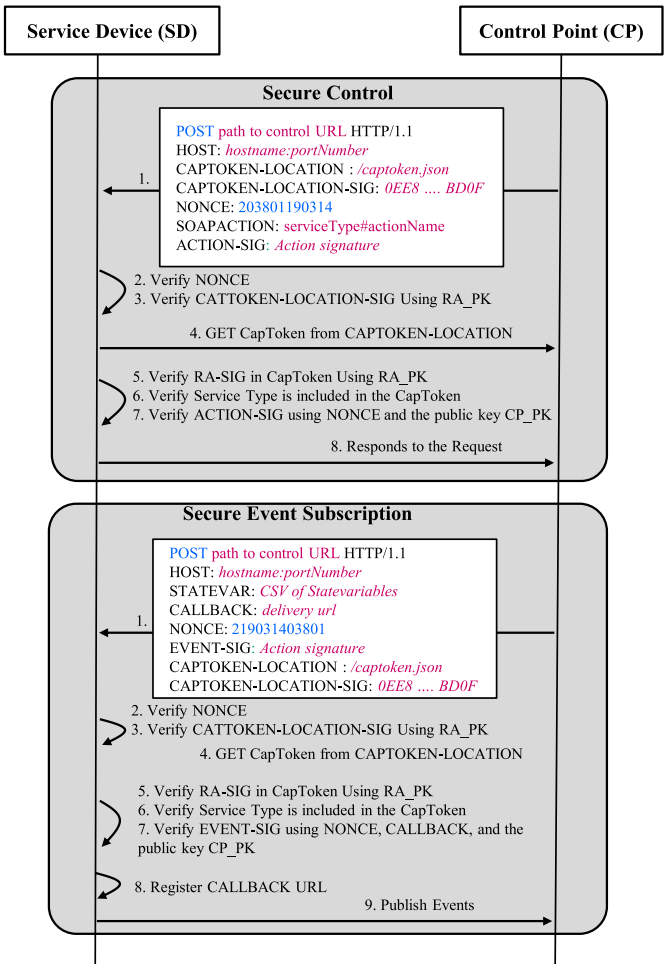

Fig. 19. Secure control and eventing.

using the RA's public key (PK_{RA}). Hence, the verification ensures the authenticity and integrity of the CapToken. The SD then retrieves the public key of the CP (PK_{CP}) and verifies the DISCOVERY-SIG as NONCE == DECRYPT(PK_{CP}, DISCOVERY-SIG) ? Authentic : Forged. Hence, the verification of DISCOVERY-SIG enables the SD to ensure the authenticity and integrity of the NONCE value.

4) Secure Control: A CP invokes actions to interact with an SD. Fig. 19 shows the interaction between CP and SD in secure control. The CP adds additional headers CAPTOKEN-LOCATION, CAPTOKEN-LOCATION-SIG, CONTROL-SIG, and NONCE to an action request. The CP copies the value of the CAPTOKEN-LOCATION-SIG from its CapToken and adds it to the request header. The field ACTION-SIG includes a signature of the CP computed as SIGN(SK_{CP}, NONCE) = ENCRYPT(SK_{CP}, Hash(NONCE)). An SD receives an action request and then validates the CAPTOKEN-URI-SIG using PK_{RA} and CAPTOKEN-LOCATION. Next, the SD retrieves the CapToken from CAPTOKEN-LOCATION. The SD validates RA-SIG of the CapToken to ensure the CapToken's authenticity and integrity. Afterward, the SD extracts the service type (e.g., serviceType1) from the SOAPACTION header and confirms that the service type is included in the CapToken. Hence, the SD ensures that the CP has the ability to consume the requested service. To this end, the SD extracts the CP's public key K_{CP} from the CapToken and validates ACTION-SIG using the NONCE and PK_{CP}. Thus, the SD authenticates the CP and ensures that the action request is not forged or spoofed.

5) Secure Eventing: An event request is just like another action on the service. Fig. 19 shows the event subscription message of SUPnP. The CP sends an event subscription request and adds CAPTOKEN-URI-SIG, CAPTOKEN-LOCATION, EVENT-SIG, and NONCE headers to the request (see Fig. 17). The CP computes the EVENT-SIG as SIGN(SK_{CP}, Hash(CALLBACK||NONCE)). An SD receives the event subscription request and validates the CAPTOKEN-URI-SIG using the PK_{RA} and ensures the authenticity and integrity of CAPTOKEN-LOCATION. The SD retrieves the CP's CapToken from CAPTOKEN-LOCATION and validates the RA-SIG to ensure the CapToken is not forged. Next, the SD validate EVENT-SIG as Hash(CALLBACK || NONCE) == DECRYPT(PK_{CP}, EVENT-SIG)? Authentic : Forged. Hence, the SD ensures that the CALLBACK URL is not forged.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

IV. Security Analysis

In this section, we present a security analysis to show that SUPnP ensures the security properties outlined in Table III under various threat scenarios that align with the threat model presented in Section II.

A. Trustworthy Capability Verification

A malicious SD or CP may attempt alter specification documents to add forged hardware capabilities or authorized services in the service list (Listings 1 and 2). However, in SUPnP, such modifications in the DSD or SAD will invalidate the CA signature (`SIG-UCA`) of the specification document. An adversary can also try to use a leaked specification document as it's own. But this will invalidate the owner's signature (`SIG-OWNER`). As part of the SUPnP protocol, a RA will identify the modification during the registration process, as the specification document verification includes verification of both `SIG-UCA` and `SIG-OWNER` using the public key of the owner (PK_p) and the public key of the CA (PK_{UCA}) (step 3 in Fig. 15).

B. Service-Device Impersonation Mitigation

As shown in Fig. 4, a malicious SD (MSD) may attempt to replay an advertisement message to impersonate a legitimate SD. The CP receives the advertisement and constructs `LOCATION URI = http://msd-ip:port//description.xml` and `CAPTOKEN-LOCATION URI = http://msd-ip:port//captoken.xml`. Next, the CP computes a hash H_{lu}= `Hash(LOCATION URI || CAPTOKEN-LOCATION URI)` and validates the `ADVERTISEMENT-SIG` as H_{lu}== `DECRYPT(PK_RA, ADVERTISEMENT-SIG)? Authentic : Forged`. Given UPnP's requirement for signatures and verification, the verification step identifies the advertisement as `Forged`, since the Malicious SD's IP address is different than the legitimate SD's IP address (step 2 of Fig. 17).

In a similar attack scenario, a malicious SD includes additional services in the device-description document (Section II-A1). However, a CP identifies the modification in the device-description document as it validates `DESCRIPTION-SIG` at step 6 of Fig. 17.

C. Control Point Impersonation Mitigation

As shown in Fig. 5, a malicious CP (MCP) may attempt to impersonate a legitimate CP by replaying a discovery message. The SD receives the discovery request and checks the `NONCE` value. If the `NONCE` is previously seen, the request will be dropped immediately (step 2 in Fig. 18). The MCP can use a new `NONCE` value to avoid an immediate drop of the request. In this case, the SD constructs a `CAPTOKEN-LOCATION URI = http://mcp-ip:port//captoken.xml`. Next, the CP computes a hash H_{lu} = `Hash(CAPTOKEN-LOCATION URI)` and validates the `CAPTOKEN-LOCATION-SIG` (step 3 of Fig. 18). Given the requirement of the capability token and signature verification in SUPnP, this attack will fail; the discovery request will be identified as `Forged`, since the mcp-ip is different than the legitimate CP's IP address.

In a similar attack scenario, the MCP may attempt to send a discovery message for a service that it is not authorized to use. The MCP forges the service type (`ST`) field of the discovery message to align with the service of interest and multicasts it in the network. The SD receives the forged discovery message and retrieves the `CapToken` of the MCP and the content of the `CapToken` using the `RA-SIG` (steps 5 and 6 in Fig. 18). After that, the SD matches the service name and type with the service name and type mentioned in the `CapToken` (step 7 in Fig. 18). Because of the SUPnP's use of capability tokens, the forged discovery message will not succeed; since the MCP's `CapToken` does not contain the name and type of the service of interest, the SD will detect the forgery and drop the request.

D. Action Authentication

A malicious CP (MCP) may attempt to use a leaked service description document to identify the information required to invoke an action, such as the control URL, the name of the actions provided by the service, the arguments of the actions (see Fig. 3). The MCP sends unauthorized action requests to the control URLs using the leaked information. The SUPnP use of capability tokens prevents against such an attack. An SD verifies the `CapToken` location and `CapToken` content using the `CAPTOKEN-LOCATION-SIG` and `RA-SIG`, respectively, (steps 3–5 in Fig. 19). The MCP cannot provide a valid, unmodified `CapToken`, so the validations fails. However, the MCP may have acquired a valid `CapToken` of its own; as such, it can pass verification of the `CAPTOKEN-LOCATION-SIG` and `RA-SIG`, but the `CapToken` does not include the name of the service of interest. The SD matches the service types mentioned in the CapToken with the service type of the action request send by the MCP (secure control step 6 in Fig. 19). As the MCP is not allowed to invoke the action on the service of interest, the service type is not found in the `CapToken`, and the SUPnP-enabled SD will therefore identify and reject the unauthorized action invocation requests.

E. Event Subscription Authentication

A malicious CP (MCP) may attempt to eavesdrop on the event subscription request from a legitimate CP. The MCP may then store the subscription request and replay it later to an SD. The SD first verifies the `NONCE` field for both of the event subscription request if the `NONCE` field (secure event subscription step 2 in Fig. 19). As the message is replayed and the `NONCE` value is previously seen the SD drops the requests immediately. The MCP can go one step further and change the `NONCE` value to avoid an immediate rejection. In SUPnP, the SD receives the request and verifies the location and content using the `CAPTOKEN-LOCATION-SIG` and `RA-SIG`, respectively. Then, the SD retrieves the public key (PK_{cp}) of the MCP from the `CapToken` and validates the EVENT-SIG as `HASH(CALLBACK ||NONCE) == DECRYPT(PK_CP, EVENT-SIG)?Authentic:Forged`. This verification

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

results in Forged during to modification of the NONCE value and the SD detects the modification of the malicious event subscription message (secure event subscription step 7 in Fig. 19).

In another attack scenario, the MCP gains unauthorized access to the description documents of an SD. Then, the MCP uses the event url field, EVE_URL (see Fig. 3) to send a malicious event subscription request. The MCP forges a CALLBACK URL in order to perform a reflection and amplification attack (Fig. 9) or DoS attack (Fig. 8). Consequently, the SD verifies the EVENT-SIG using the hash of CALLBACK URL and the NONCE value as stated in the last attack scenario. The verification of the EVENT-SIG fails as the MCP modifies the CALLBACK URL. The SD detects malicious modification of the CALLBACK URL by the MCP and rejects the event subscription request.

V. Experiment and Evaluation

In this section, we present an experimental evaluation of the SUPnP protocol through simulation to demonstrate the feasibility of implementing the approach in an IoT network containing resource-constrained devices. Our evaluation compares the performance of our SUPnP solution to the UPnP protocol, under typical operation and in various attack scenarios.

A. Experimental Model

To demonstrate the feasibility of implementing SUPnP, our evaluation focuses on metrics that are relevant to the performance and scalability in networks of the resource-constrained IoT devices. Specifically, we compare SUPnP and UPnP in terms of energy consumption in resource-constrained IoT devices, network throughput, and latency while performing service registration, advertisement, and discovery operations. In our experiments, we configured attacking participants such a way that they make use of CP impersonation, discovery message forgery and discovery message flooding attacks against an SD.

1) Network Setup: We emulate a real-world UPnP network, configured as shown in Fig. 20. Using the Cooja [33] simulator, we create a IoT network which consists a number of simulated Z1 [34] devices, shown in the gray box. One of the Z1 devices acts as a 6LoWPAN border router (represented as the blue node) using Contiki's 6LBR package [35]. In addition to the simulated IoT devices, the experimental network includes real physical devices: the desktop computer, supporting the RA; a wireless access point (WAP), connected through the Ethernet (eth0) interface with the desktop (RA); an Android smartphone, connected to the WAP via WiFi interface; an laptop, connected to the WAP through WiFi interface. The simulated network is connected to the desktop (RA) through Serial LineIP (tun0) [36] using the Tunslip utility [37] provided in Cooja.

2) Device Configurations: The HW and SW configurations of the devices used in the experimental setup is shown in Table IV. All of the devices except the Android smartphone, were configured either as a CP or an SD. The Android smartphone was configured both as CP and SD. All the Z1 IoT devices except one are configured as CPs. One Z1 device was configured as an SD. The SDs (the Z1 device and the smartphone) were configured to provide three services each containing three actions. The CPs sent discovery message to find the services provided by the SDs in an interval I, where I$\in$ {1000 ms, 2000 ms, 3000 ms}. We varied the interval I, to show the effects on the discovery message frequency in our evaluation metrics. We introduced nodes as attacking CPs, which send the discovery requests more frequently than the normal nodes; we varied the number of malicious CPs, $N_{\text{malicious}} \in \{1, 2, 4\}$ to show the effects of different number of attackers in the UPnP network. All devices in the network use UDP as the transport-layer protocol to exchange UPnP discovery and advertisement messages. The desktop implemented all the RA services in a JAVA-based implementation.

TABLE IV
Description of Experimental Devices

Device	HW/SW Configuration	Role
Desktop	Model: Dell Precision 3630 CPU:Intel Core i5-9500, 3.0Ghz OS: Ubuntu 16.04 RAM: 16 GB ROM: 512 GB	RA
Z1	CPU: 16 MHz MSP430F2617 OS: Contiki RAM: 8 KB ROM:92 KB	Control Point / Service device
Smart Phone	Model: Samsung S8 CPU: Octa-core (2.3 GHz Quad + 1.7 GHz Quad) OS: Android RAM: 4 GB ROM: 64 GB	Control Point & Service Device
Laptop	Model: Lenovo Legion Y520 CPU: 7the Generation Core-I7 OS: Ubuntu 16.04 RAM: 16 GB ROM: 512 GB	Control point

3) Data Collection and Prepossessing: We used the RPL UDP [38] example of the Contiki to implement the UPnP SD and CP in Z1 IoT devices. We measured the energy consumption of the Z1 devices using the Powertrace library [39] which uses the Contiki's energy APIs to measure the power consumption of CPU and radio transceiver. To measure the throughput we leveraged the Contiki logging [40] mechanism to log the timing and size of the data received by a Z1 device. We implemented a Python-based Contiki Log Parse to extract the data readings from the Contiki raw logs.

We developed an Android application for the smartphone that sends discovery and advertisement messages. We collect the timing information and size of the data received from the Android log and parse it using a Python engine. Similarly, in the laptop, we implemented a Java-based UPnP CP to send discovery messages and log the data into a log file for further analysis.

B. Energy Consumption Analysis

In this experiment, we analyze the effect of the UPnP discovery flooding attack (see Fig. 6) on an IoT SD's energy consumption using SUPnP and compare it with basic UPnP.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

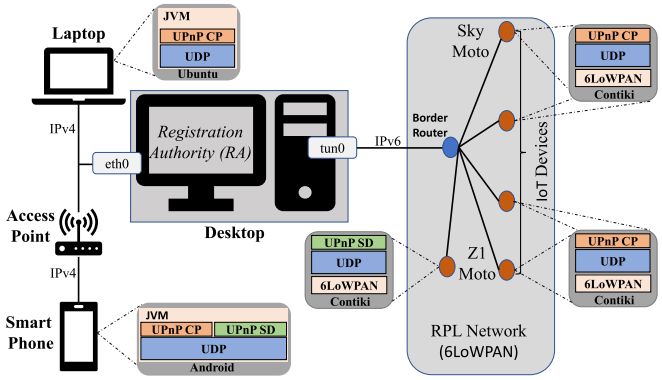

Fig. 20. Experimental UPnP network.

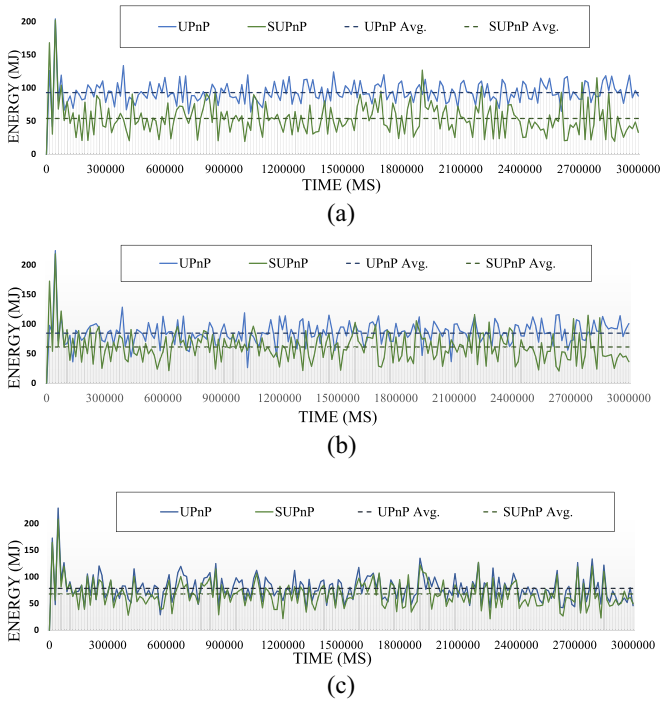

Fig. 21. Average energy consumption UPnP versus SUPnP under different attack interval. (a) Attack interval 1000 ms. (b) Attack interval 2000 ms. (c) Attack interval 3000 ms.

In the first part of our energy analysis, we emulated a real-world UPnP scenario, where four CPs searching for the same service in the network, with one of these CPs acting as an adversary. A fifth Z1 device acts as an SD that provides the service requested by the CPs. The environment is reflective of the number of devices in a real-world IoT application for smart homes or a similar scenario. The malicious CP sent a forged discovery message, where the CAPTOKEN_URI_SIG is forged. The cooperative CPs sent the discovery requests periodically at a reasonable interval (10 000 ms) while the adversary sends discovery request more frequently. Fig. 21 shows that the average energy consumed under SUPnP is lower than UPnP.

In the next part of our experiment, we increase the number of attackers. With 1, 2, and 4 attackers SUPnP gains 36, 19.5, and 10 mJ of energy on average, respectively. This is

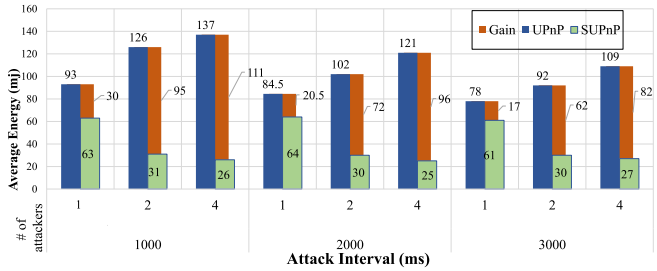

Fig. 22. Gain in energy consumption SUPnP versus UPnP (under attack).

because, with SUPnP, the SD detected and rejected the malicious requests, reducing the energy consumed by eliminating the transmission of response messages to forged discovery requests. Although some computational energy is required to verify an authentic discovery request, it is substantially less than the energy required to reply to the message. It follows that as the frequency of malicious requests increases, the average energy gain by SUPnP increases as well. Similarly, as shown in Fig. 22, the energy gain for SUPnP increases as the number of malicious CPs increases or the frequency of the malicious request increases. For example: with 2000-ms attack frequency and four attacking nodes, SUPnP achieves an energy gain of 96 mJ compared to UPnP. If the attack frequency is lower to 1000 ms with four attacking nodes, the said gains becomes 111 mJ.

Under normal network scenarios, SUPnP incurs some additional overhead due to the extra operations required in the discovery phase for authentication. Research shows that selecting good intervals for advertisement and discovery is not easy considering network congestion, power consumption, and user experience [41], [42]. In our experiments, we set advertisement interval for a normal SD to 15 min, which is half of the default advertisement time to live, CACHE-CONTROL (see Fig. 2) field based on the recommendation of UPnP specification [30]. To show the overhead of SUPnP under normal network scenario (without any malicious CPs), we compare the energy consumption of SUPnP with basic UPnP under three scenarios, executed over a 60-min time window: 1) no CP is requesting discovery and the SD periodically advertises a service according in 15-min interval; 2) four CPs are issuing service discovery requests every 2 min; and 3) eight CPs are requesting service discovery every 2 min. We repeated the experiment ten times and average the result for better consistency. Fig. 23 shows the experimental results. The data clearly show that the average energy consumption for SUPnP is negligible. For example, when eight nonmalicious clients asked for legitimate discovery, SUPnP causes only around a 3% increase in normal energy consumption. The minimal energy overhead indicates that SUPnP is applicable for resource constrained UPnP-enabled IoT devices.

C. Throughput Analysis

Usually, as part of an IoT system, SDs are deployed in lossy networks (e.g., a 6LoWPAN network) like our experimental setting. With a large number of requests, the rate of the request

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

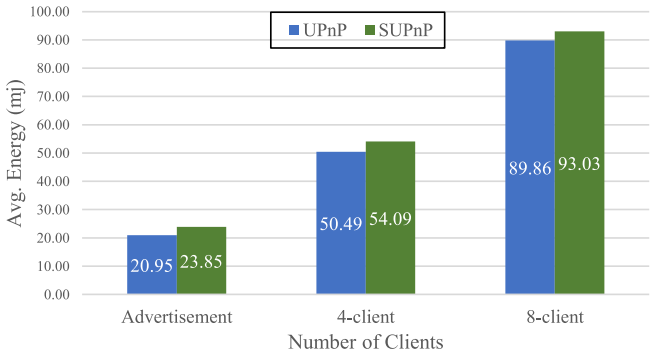

Fig. 23. Energy consumption comparison SUPnP versus UPnP (normal scenario).

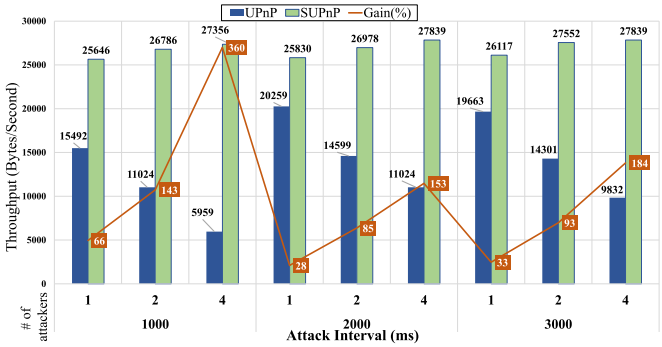

Fig. 24. Throughput comparison SUPnP versus UPnP (attack scenario).

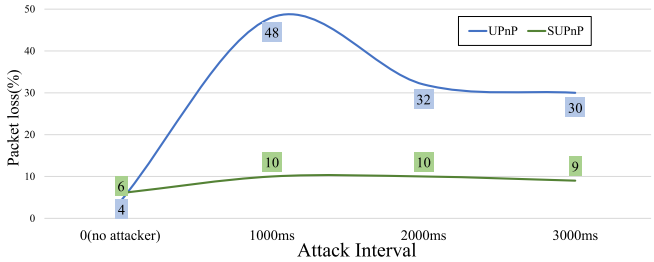

Fig. 25. RDR SUPnP versus UPnP (attack scenario).

drop increases and the network throughput decreases. The malicious CPs can leverage this limitation of lossy networks by sending a large number of UPnP discovery packets toward an SD, resulting in a decrease in throughput and eventually leading to denial of service (see Section II). In this experiment, we assessed how the throughput was impacted for requests made by a nonmalicious CP to a nonmalicious SD when numerous discovery flooding attacks are employed by other malicious CPs in an IoT network. Specifically, we measure

$$\text{Throughout} = \frac{\text{total nonmalicious bytes received by a SD}}{\text{Total End-to-end delay to send the byte}}. \tag{1}$$

We created a real-life scenario by considering a smartphone as CP and tried to discover a service from an IoT SD. Here, the smartphone acted as the nonmalicious CP connected to the UPnP network via the WiFi interface and the IoT SD that provides the targeted service is located in the lossy network (see Fig. 20). Meanwhile, some other malicious CPs flood the lossy network, targeting the service provider SD. We varied the number of malicious CPs to observe the affect in the network throughput with an increasing number of active attackers deploying discovering flooding attack. We also changed the frequency of the malicious discovery requests to see the impact on throughput with a large number of malicious requests.

In the experimental scenario, the smartphone sent discovery requests to an SD with an interval of 10 000 ms. Different number of malicious CP deployed discovery attack using different time intervals. In measuring the throughout we only considered the discovery requests by nonmalicious CP as shown in (2). The effect of throughput under different attack scenarios are shown in Fig. 24. As shown, as the number of attackers increases, the throughput of the SDs decreases for basic UPnP. In contrast, the throughput does not change substantially for SUPnP. This is because, with SUPnP validation mechanisms, the SD detects the malicious requests and drops them without replying, which can reduce the potential for network congestion. For example, SUPnP gains almost 153% in network throughput with four attacking CPs and with attack frequency 2000 ms. As such, SUPnP provides significant performance gains in terms of throughput in discovery flooding attack scenarios, with minimal reductions in throughput under normal scenarios, as compared to UPnP.

1) Request Drop Rate: As we mentioned earlier, IoT applications are often supported in lossy networks (e.g., 6LoWPAN). While there is large network traffic the requests are often dropped in lossy networks. Exploiting this property, a malicious CP can deploy discovery flooding attacks to cause a congestion in the lossy network, which, in turn, increase the rate of legitimate UPnP request drop. In this experiment, we measured the RDR when the UPnP network is going though a flooding attack. Similar to the throughput, in measuring the RDR, we considered a nonmalicious CP making request to a nonmalicious SD, while another malicious CP is flooding the network with discovery message. We used the below equation to measure the RDR

$$\text{RDR} = \frac{\text{Total nonmalicious request sent}}{\text{Total nonmalicious request received}}. \tag{2}$$

Here, the nonmalicious CP sent the discovery messages periodically at a reasonable interval (10 000 ms) and the malicious CP sent the discovery message in different more frequent intervals. We present the result of our experiment in Fig. 25. We performed the experiments 100 times and average the results for better accuracy. As shown in Fig. 25 the value of *RDR* in SUPnP was lower than UPnP in all different attack frequencies. Although, without any attacking CP, *RDR* was very similar for both SUPnP and UPnP. SUPnP discards the malicious discovery requests without responding, which limits the impact of network congestion caused by the attack, and therefore in SUPnP the *RDR* is lower. On the other hand, in UPnP, the SD had to reply all the discovery messages, including the malicious ones, which, in turn, increase the *RDT* value.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

TABLE V
SECURITY PROPERTIES COMPARISON OF SUPNP WITH RELATED WORK

Scheme	Capability Verification		Service-device Impersonation		Control-point Impersonation		Malicious Action		Malicious Event Subscription	
	Detection	Prevention	Detection	Prevention	Detection	Prevention	Detection	Prevention	Detection	Prevention
UPnP	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
UPnP-UP [43]	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
U-PoT [44]	✗	✗	✓	✗	✓	✗	✓	✗	✓	✗
KUPnP [45]	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
Guo et al [46]	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Pehkonen et al. [47]	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗
Islam et al [48]	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
SUPnP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

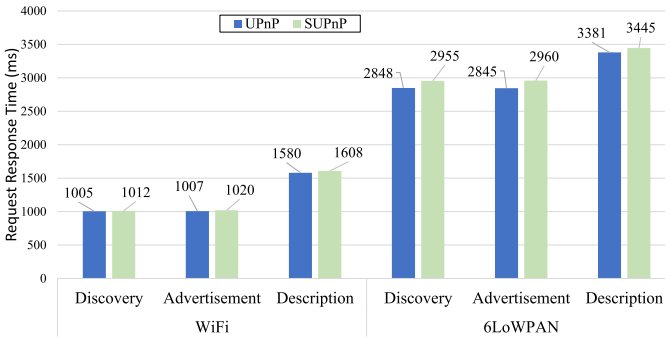

Fig. 26. RRT UPnP versus SUPnP.

Our experimental results showed that SUPnP improved the service quality by decreasing the RDR value, which is desired in lossy IoT networks.

2) Request Response Time: We measure the time required for an SD to reply a legitimate UPnP request using the following equation:

$$\text{RRT} = T1 + T2 + T3. \tag{3}$$

Here, RRT = Request Response Time, $T1$ = the time required to transmit a message from the CP to the SD, $T2$ = the time to verify the message (CapToken verification), and $T3$ = the time to prepare and send a response.

We conducted this experiment in both WiFi (data-rate 600 Mb/s) and 6LoWPAN (data-rate 128 kb/s) medium as the communication links. In both cases, a nonmalicious CP send a discovery message to a nonmalicious SD. In WiFi medium, the laptop acted as the CP and the smartphone acted as the SD. Similarly in 6LoWPAN medium a Z1 device acted as the SD and another Z1 device acted as the CP. Fig. 26 shows the result of our experiments. We conducted 100 iterations of the experiment report the mean result. Our result shows that the delay incurred by SUPnP in both traditional UPnP networks (Wifi medium) and IoT network (6LoWPAN) is very low compare to the basic UPnP: 1.33% for WiFi and 3.15% for 6LoWPAN. That indicated that SUPnP has a small execution time overhead due to the computation costs for capability verification. Thus, our experimental result indicate the feasibility of SUPnP in terms of the service response time in service oriented UPnP IoT networks.

VI. RELATED WORKS AND COMPARATIVE DISCUSSION

In this section, we present the prior efforts to secure the IoT devices and UPnP network. The security risks of IoT-integrated cyber–physical systems are well explored. Although interest continues to grow in the development of approaches grounded in artificial intelligence and machine learning to detecting security threats in IoT networks [49]–[53], we focus on developing a UPnP-enabled solution that addresses identified threats. Here, we present efforts related to devising secure UPnP protocols. Table V presents a summarized comparison of how existing UPnP protocols address the security properties embodied by our SUPnP solution.

U-PoT [44] takes a different approach in securing UPnP, making use of honey pots to mitigate the attacks on the discovery and description phase of UPnP. In this approach, a honey pot is generated by automatically creating an emulated UPnP device from a UPnP device description document. While this approach may show promise in terms of detecting malicious actions, event subscriptions, and replay attacks in the UPnP IoT network, it is costly and difficult to deploy. Moreover, unlike SUPnP, U-Pot does not prevent unauthorised discovery and access of the UPnP services.

UPnP-UP [43] is an extension of the UPnP that enables a multilevel security protocol that can be used for user authentication and authorization and to achieve interoperability among the available services in a network. UPnP-UP provides a network manager with ability to define access control policies using a well-defined user interface. In addition, this provides a flexible way to select the security properties based on the need of pervasive networks (such as residence environment, commercial environment, and secure environment). However, UPnP-UP only focuses on the service requests and invocations of actions which is associated with the eventing and control phase of UPnP. Unlike SUPnP, it does not provide any security mechanisms that address vulnerabilities in the discovery and advertisement phase of UPnP. Moreover, this scheme cannot verify the capability of the UPnP devices before joining the network, as SUPnP does.

Guo and Li [46] designed an UPnP key management scheme that based on the group signature algorithm. The scheme includes the member join, signature verification, and secure communication among the group members. But the assumption of this work is a small amount devices need

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

interoperability which is not suitable for IoT scenarios. If the group consists a large amount of IoT devices group signature-based approach is not be scalable, thus cannot be deployed in large scale IoT networks. In contrast, SUPnP does not require to maintain any information in any centralized entity, thus does not have scalability issue. Besides, this work focuses only on the securing action invocation request, does not include securing discovery, advertisement and eventing as SUPnP does.

KUPnP [45] proposes a UPnP extension based on the Kerberos service [54] to secure the UPnP devices and control points by introducing a key distribution center (KDC) as a central manager to handle authentication between devices and key management. However, the KDC needs to centrally maintain a database consisting all the secret keys of the control points and service devices that causes a scalability issue with large number of participants. SUPnP addresses this issues by assigning a CapToken to the devices, which is used in authentication process, so there is no need to maintain a centralized database. Finally, KUPnP does not defend against impersonations attacks by an adversary changing the service description file and removing the service required to perform Kerberos-based authentication. In contrast, SUPnP detects and prevents impersonation attacks caused by modification the description files.

Pehkonen and Koivisto [47] proposed a secure UPnP network architecture using transport-layer security (TLS) to secure all TCP traffic, which carries most of UPnP discovery and advertisement messages. However, this work only focuses on the discovery architecture of UPnP, unlike SUPnP it does not provide any secure solution for UPnP control and eventing mechanism.

Islam *et al.* [48] proposed a simpler access control system without incorporating any complex authentication procedures by adopting access control list (ACL) to defend DLNA supported devices from unwanted control points. This work uses a particular field in requests or responses to identify a control point and manages a list of access rules. That approach is a good candidate to protect the UPnP devices from unauthorized control points due the lightweight nature of the authentication procedures. But maintain a large number of access rules is always challenging, even not feasible in large IoT networks. SUPnP does to need to store any access rules, so it has no scalability issue and suitable for IoT networks. Moreover, this work focuses on the control phase of UPnP, unlike SUPnP, does not include the security concerns of discovery, advertisement and eventing phases.

VII. Concluding Remarks

In this work, we conducted a security analysis of the UPnP protocol, resulting in the identification of security vulnerabilities in service discovery, advertisement, control, and eventing phases. Our analysis highlighted how an adversary can launch service device impersonation, control-point impersonation, denial of service, and resource exhaustion attacks when UPnP is deployed in service-oriented systems in IoT networks. To address these issues, we introduced a capability-based security scheme SUPnP, which enables both service devices and control points to verify that service devices are capable of and authorized to support and use services. Our experimental evaluation of our implementation of the SUPnP protocol highlighted the feasibility of deploying the proposed scheme in real-world IoT scenarios; SUPnP shows significant reductions in energy consumption and substantial increases in throughput under several attack scenarios compared to UPnP, with reasonable tradeoffs in energy consumption and negligible increases in latency under normal scenarios.

As shown in our analysis, the SUPnP defends against various critical attacks and ensures secure service advertisement and discovery. It prevents impersonations and DoS attacks and helps to avoid buffer overflows. Nevertheless, the SUPnP does not consider the access control from the individual user's perspective and is, therefore, unable to restrict the network access based on their roles. Role-based access control is highly important to facilitate secure access in large organizations such as hospitals, where hundreds of users and thousands of permissions are required in a hierarchical manner. However, the conventional access control methods cannot be applied directly to UIoT settings due to the resource constraints and distributed architecture of the relevant systems. Therefore, the investigation for the lightweight and scalable solutions for secure access control for UPnP-enabled IoT can be worthy research. We can also investigate machine learning-based and deep learning-based solutions exploiting the fusion of statistical and contextual user request behaviors to execute a large-scale secure access for UPnP-enabled IoT networks.

References

[1] S. Bagchi *et al.*, "New frontiers in IoT: Networking, systems, reliability, and security challenges," 2020. [Online]. Available: arXiv:2005.07338.

[2] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. Kwak, "The Internet of Things for health care: A comprehensive survey," *IEEE Access*, vol. 3, pp. 678–708, 2015.

[3] Y. A. Qadri, A. Nauman, Y. B. Zikria, A. V. Vasilakos, and S. W. Kim, "The future of healthcare Internet of Things: A survey of emerging technologies," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1121–1167, 2nd Quart., 2020.

[4] Z. Bi, L. D. Xu, and C. Wang, "Internet of Things for enterprise systems of modern manufacturing," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1537–1546, May 2014.

[5] L. Yushi, J. Fei, and Y. Hui, "Study on application modes of military Internet of Things (MIoT)," in *Proc. IEEE Int. Conf. Comput. Sci. Autom. Eng. (CSAE)*, vol. 3, 2012, pp. 630–634.

[6] F. Zhu, Y. Lv, Y. Chen, X. Wang, G. Xiong, and F. Wang, "Parallel transportation systems: Toward IoT-enabled smart urban traffic control and management," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 10, pp. 4063–4071, Oct. 2020.

[7] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang, "IoTCandyJar: Towards an intelligent-interaction honeypot for IoT devices," in *Proc. Black Hat conf.*, pp. 1–11, Aug. 2017.

[8] M. Jeronimo and J. Weast, *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Hillsboro, OR, USA: Intel Press, 2003.

[9] *Security Flaws in Universal Plug and Play: Unplug, Don't Play*. sAccessed: Feb. 22, 2020. [Online]. Available: https://blog.rapid7.com/2013/01/29/security-flaws-in-universal-plug-and-play-unplug-dont-play/

[10] H. Park, M. Choi, E.-H. Paik, and N. Kim, "Interoperability model for devices over heterogeneous home networks," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1185–1191, Aug. 2009.

[11] H. Moore, "Security flaws in universal plug and play," Rapid7, Boston, MA, USA, White Paper, Jan. 2013.

[12] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security evaluation of home-based IoT deployments," in *Proc. IEEE Symp. Security Privacy (SP)*, 2019, pp. 1362–1380.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

[13] G. Kayas, M. Hossain, J. Payton, and S. R. Islam, "An overview of upnp-based IoT security: Threats, vulnerabilities, and prospective solutions," in *Proc. 11th IEEE Annu. Inf. Technol. Electron. Mobile Commun. Conf. (IEMCON)*, 2020, pp. 0452–0460.

[14] Z. A. Khan and U. Abbasi, "Reputation management using honeypots for intrusion detection in the Internet of Things," *Electronics*, vol. 9, no. 3, p. 415, 2020.

[15] *Shodan*. Accessed: Jun. 22, 2020. [Online]. Available: https://www.shodan.io/

[16] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMAP: Fast Internet-wide scanning and its security applications," in presented at the 22nd USENIX Security Symp. (USENIX) Security), 2013, pp. 605–620.

[17] M. Antonakakis *et al.*, "Understanding the mirai botnet," in *Proc. 26th USENIX Security Symp.*, 2017, pp. 1–19. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[18] *Massive Qbot Botnet Strikes 500,000 Machines Through Wordpress*. Accessed: Feb. 22, 2020. [Online]. Available: https://www.infosecurity-magazine.com/news/massive-qbot-strikes-500000-pcs/

[19] Y. Cadirci. (2020). *CallStranger*. Accessed: Apr. 14, 2020. [Online]. Available: https://github.com/yunuscadirci/CallStranger/blob/master/CallStranger

[20] G. Kayas, M. Hossain, J. Payton, and S. R. Islam, "VSDM: A virtual service device management scheme for upnp-based IoT networks," in *Proc. 11th IEEE Annu. Ubiquitous Comput. Electron. Mobile Commun. Conf. (UEMCON)*, 2020, pp. 426–433.

[21] I. Butun, P. Österberg, and H. Song, "Security of the Internet of Things: Vulnerabilities, attacks, and countermeasures," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 616–644, 1st Quart., 2020.

[22] H. Song, G. Fink, and S. Jeschke, *Security and Privacy in Cyber-Physical Systems*. Hoboken, NJ, USA: Wiley, 2017.

[23] X. Jin, R. Sandhu, and R. Krishnan, "RABAC: Role-centric attribute-based access control," in *Proc. Int. Conf. Math. Methods, Models, Archit. Comput. Netw. Security*, 2012, pp. 84–96.

[24] S. Lawrence, "Formal models of capability-based protection systems," *IEEE Trans. Comput.*, vol. C-30, no. 3, pp. 172–181, Mar. 1981.

[25] C. R. Landau, "Security in a secure capability-based system," *Oper. Syst. Rev.*, vol. 23, no. 4, pp. 2–4, Mar. 1989.

[26] D. Servos and S. L. Osborn, "Current research and open problems in attribute-based access control," *ACM Comput. Surveys*, vol. 49, no. 4, pp. 1–65, Jan. 2017. [Online]. Available: http://doi.acm.org/10.1145/3007204

[27] M. N. Aman, S. Taneja, B. Sikdar, K. C. Chua, and M. Alioto, "Token-based security for the Internet of Things with dynamic energy-quality tradeoff," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2843–2859, Apr. 2019.

[28] S. Malani, J. Srinivas, A. K. Das, K. Srinathan, and M. Jo, "Certificate-based anonymous device access control scheme for IoT environment," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9762–9773, Dec. 2019.

[29] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse," in *Proc. NDSS*, 2014, pp. 1–15.

[30] Open Connectivityn. (2020). *UPnP Device Architecture 2.0*. Accessed: Apr. 14, 2020. [Online]. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf

[31] OCF. (2020). *UPNP Certification*. Accessed: Apr. 14, 2020. [Online]. Available: https://openconnectivity.org/certification/upnp-certification/

[32] UPnP, Forum. "Internet Gateway Device (IGD) Standardized Device Control Protocol V1.0." 2001. Accessed: Feb. 23, 2021. [Online]. Available: https://www.semanticscholar.org/paper/Internet-Gateway-Device-(IGD)-Standardized-Device-UPnP/cd684cdf64a49c43c69d3d8aff427752310a778e

[33] Contiki. (2019). *An Introduction to Cooja*. Accessed: Jul. 14, 2020. [Online]. Available: https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja

[34] (2018). *Zolertia*. Accessed: Jul. 14, 2020. [Online]. Available: https://github.com/Zolertia/Resources/wiki/The-Z1-mote

[35] Cetic. (2020). *6LBR: A 6LoWPAN Border Router Based on the Contiki Operating System*. Accessed: Apr. 14, 2020. [Online]. Available: https://github.com/cetic/6lbr

[36] J. Romkey, "Nonstandard for transmission of IP datagrams over serial lines: Slip," IETF, RFC 1055, 1988.

[37] ANRG. (2020). *RPL Border Router*. Accessed: Apr. 14, 2020. [Online]. Available: http://anrg.usc.edu/contiki/index.php/RPL_Border_Router

[38] Contiki. (2020). *RPL-UDP*. Accessed: Apr. 14, 2020. [Online]. Available: https://github.com/contiki-os/contiki/tree/master/examples/ipv6/rpl-udp

[39] Contik. (2017). *Contiki Apis for Measuring Energy Consumption*. [Online]. Available: http://contiki.sourceforge.net/docs/2.6/a00452_source.html

[40] Contiki. (2020). *Contiki Logging*. Accessed: Nov. 14, 2020. [Online]. Available: https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-Logging

[41] K. Mills and C. Dabrowski, "Adaptive jitter control for UPNP *m*-search," in *Proc. IEEE Int. Conf. Commun. (ICC)*, vol. 2, 2003, pp. 1008–1013.

[42] Y.-L. Liong and Y. Ye, "Effect of upnp advertisements on user experience and power consumption," in *Proc. 2nd IEEE Consum. Commun. Netw. Conf. (CCNC)*, 2005, pp. 91–97.

[43] T. M. Sales, L. M. Sales, H. O. Almeida, A. Perkusich, and M. A. de Sales, "Multilevel security in UPNP networks for pervasive environments," *IEEE Trans. Consum. Electron.*, vol. 59, no. 1, pp. 144–152, Feb. 2013.

[44] M. A. Hakim, H. Aksu, A. S. Uluagac, and K. Akkaya, "U-POT: A honeypot framework for UPNP-based IoT devices," in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf. (IPCCC)*, 2018, pp. 1–8.

[45] H. Zhu and Y. Zhu, "A kerberos-based upnp extention for secure home networks," in *Proc. 4th Int. Conf. Comput. Eng. Technol.*, 2012, pp. 104–108.

[46] X. Guo and J. Li, "Secure UPNP services based on group signature algorithm," in *Proc. 3rd Int. Conf. Comput. Sci. Netw. Technol.*, 2013, pp. 951–955.

[47] V. Pehkonen and J. Koivisto, "Secure universal plug and play network," in *Proc. IEEE 6th Int. Conf. Inf. Assurance Security*, 2010, pp. 11–14.

[48] M. Z. Islam, M. M. Hossain, S. Haque, J. Lahiry, S. A. Bonny, and M. N. Uddin, "User-agent based access control for DLNA devices," in *Proc. IEEE 6th Int. Conf. Knowl. Smart Technol. (KST)*, 2014, pp. 7–11.

[49] Z. Lv, L. Qiao, J. Li, and H. Song, "Deep learning enabled security issues in the Internet of Things," *IEEE Internet Things J.*, early access, Jul. 9, 2020, doi: 10.1109/JIOT.2020.3007130.

[50] Z. Lv, S. Zhang, and W. Xiu, "Solving the security problem of intelligent transportation system with deep learning," *IEEE Trans. Intell. Transp. Syst.*, early access, Mar. 20, 2020, doi: 10.1109/TITS.2020.2980864.

[51] Z. Lv and F. Piccialli, "The security of medical data on Internet based on differential privacy technology," *ACM Trans. Internet Technol.*, to be published.

[52] Z. Lv, L. Qiao, and H. Song, "Analysis of the security of Internet of multimedia things," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 16, no. 35, p. 97, 2020.

[53] Z. Lv, D. Chen, R. Lou, and H. Song, "Industrial security solution for virtual reality," *IEEE Internet Things J.*, early access, Jul. 1, 2020, doi: 10.1109/JIOT.2020.3004469.

[54] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 33–38, Sep. 1994.

Golam Kayas is currently pursuing the Ph.D. degree with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA.

He served the Samsung R&D Institute, Dhaka, Bangladesh, as a Senior Software Engineer from June 2013 to August 2017. His research interests include security and privacy of Internet of Things, pervasive computing, and cloud computing.

Mahmud Hossain received the Ph.D. degree in computer science from the University of Alabama at Birmingham (UAB), Birmingham, AL, USA, in April 2018.

He is a Senior Cyber-Security Researcher with the Cyber Engineering Department, Visa, Inc., San Francisco, CA, USA. He was a Research Assistant with the Secure and Trustworthy Computing Lab, UAB. He served Samsung R&D Institute, Dhaka, Bangladesh, as a Lead Engineer with the Department of Solution Lab from June 2010 to August 2014. His research interests include security and privacy of Internet of Things, embedded systems, cloud computing, applied cryptography, digital forensics, and blockchain networks.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.

Jamie Payton (Member, IEEE) received the D.Sc. in computer science and engineering from Washington University in St. Louis, St. Louis, MO, USA, in 2006

She is an Associate Professor and the Chair of the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. She is also the Director of the STARS Computing Corps, a national alliance of over 50 colleges and universities with the mission to broaden participation in computing. STARS has engaged over 2500 college students in service learning projects with regional K12 schools, industry, and community partners to inform, engage, and prepared over 135 000 future students for entry and success in college computing programs. Her research is centered around the design of pervasive and wearable computing systems.

S. M. Riazul Islam received the Ph.D. in information and communication engineering from Inha University, Incheon, South Korea, in 2012.

He is an Assistant Professor with the Department of Computer Science and Engineering, Sejong University, Gwangjin, South Korea, since 2017. From 2014 to 2017, he worked with the Wireless Communications Research Center, Inha University, Incheon, South Korea, as a Postdoctoral Fellow. From 2016 to 2017, he was also affiliated with Memorial University, St. John's, NL, Canada, as a Postdoctoral Fellow. From 2005 to 2014, he was with the University of Dhaka, Dhaka, Bangladesh, where he was an Assistant Professor and a Lecturer with the Department of Electrical and Electronic Engineering. In 2014, he worked with the Samsung R&D Institute Bangladesh, Dhaka, as a Chief Engineer with the Department of Solution Lab for advanced research. His research interests include wireless communications, Internet of Things, and applied artificial intelligence.

Authorized licensed use limited to: Temple University. Downloaded on September 28,2021 at 18:34:37 UTC from IEEE Xplore. Restrictions apply.