Applying Internal Interface Diversification to IoT Operating Systems

Lauri Koivunen
University of Turku
Turku, Finland
Email: lamkoi@utu.fi

Sampsa Rauti
University of Turku
Turku, Finland
Email: sjprau@utu.fi

Ville Leppänen
University of Turku
Turku, Finland
Email: ville.leppanen@utu.fi

***Abstract*—Internet of Things (IoT) currently covers billions of devices with identical internal software interfaces. This software monoculture exposes the systems to the same security vulnerabilities. Internal interface diversification, by introducing diverse and unique interfaces on each device, is a solution for this problem. In this paper, we discuss interface diversification in the context of IoT operating system. We study internal interfaces of ten different operating systems to gauge the feasibility of diversification in IoT environment. This discussion brings us closer to practical diversification implementations for IoT operating systems.**

I. Introduction

The Internet of Things (IoT) is a network of interconnected physical devices and other objects [33]. These ”things” are incorporated with software, sensors and electronics. The network connection allows the devices to gather and exchange information. The IoT can be seen as a global infrastructure offering advanced services with the aim of making human life more effortless and comfortable. Today, IoT is used by various public and private sectors from health care to manufacturing and industrial applications. IoT is continuously growing. More and more new devices are connected to Internet daily. Estimates vary, but according to Gartner, there are over 6 billion ”things” in use right now and there will be about 20 billion of them by 2020 [16].

The recommendation by International Telecommunication Union (ITU) states that ”IoT makes full use of things to offer services to all kinds of applications, whilst ensuring that security and privacy requirements are fulfilled.” [33] However, security and privacy are still huge challenges for this new environment. According to a report by HP, at least 70 percent of IoT devices are vulnerable to exploits that may cause harm to the users [15]. There is a great demand for improving software security in this environment. In our previous work, we have proposed obfuscation and diversification approaches as potential solutions for alleviating the risk of malicious attacks [19]. This paper continues our work on this front.

In this paper, we propose applying interface diversification to different internal interfaces and software layers of operating systems in the context of IoT. In order to gauge the feasibility of diversification methods in different IoT environments, we examine ten existing IoT operating systems and their diversifiable internal interfaces. These novel ideas we present are a step towards practical implementations of diversification approaches for IoT operating systems.

The rest of the paper is structured as follows. Section 2 discusses the importance of operating system security in the IoT and explains the concept of interface diversification. Section 3 presents our review of existing IoT operating systems and their diversifiable internal interfaces that could be targets for diversification. Section 4 discusses the benefits and drawbacks of interface diversification. Section 5 concludes the paper.

II. Interface Diversification

Diversification makes interfaces unique, and an adversary is no longer able to exploit a large number of devices with the same exploit. In order to launch a successful attack, the attacker would need to design various versions of the exploit, which is laborious and costly. Figure 1 illustrates diversification of interfaces. Unique versions of an interface are created and distributed to users. Even if the adversary manages to compromise one version of the interface, the other replicas are still safe. Obfuscation techniques [13] such as changing the function names of an interface are used to accomplish diversification. The functionality of code remains the same, but the internal interfaces of generated unique software versions are different.

When an interface has been diversified, only the trusted programs in the system can use the interface, because they know the diversification secret. In other words, the diversification is propagated to the trusted programs so that they know the implementation details of the diversified interfaces they need to make use of. It is apparent that diversification as a security mechanism bears a resemblance to encryption. However, diversified code is still executable as such whereas an encrypted program is not.

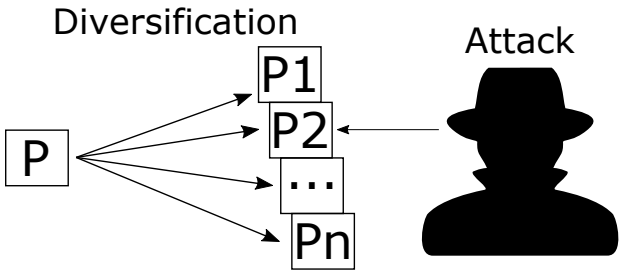

Figure 1. Diversification creates multiple unique instances from the original interface.

It is worth noting that diversification does not remove the security holes present in software. However, the malware injected into the system using the vulnerability will not work because it would need to access services through the diversified secret internal interfaces.

This is why diversification is a useful mechanism for protecting the systems against the dangers of prevailing software monoculture (the same well-known internal interfaces in every copy of the operating system). It is also useful in settings like IoT, where the system is not regularly updated and vulnerabilities are not patched.

The term *interface* is used quite broadly in this paper. Here, interface doe not only refer to ordinary interfaces provided by software components. Instead, we also see e.g. or memory addresses (of services/resources) and commands of a language as interfaces that can potentially be targets for diversification. From this perspective, an interface is anything that can be used to access the resources of the system.

As an example, a practical case of interface diversification could be changing the mapping of the system call numbers used in an operating system. System calls are used by applications to request a service from the operating system and gain access to computer's critical resources. It therefore makes sense to diversify them to prevent the adversary from using system calls.

We also make a distinction between external and internal interfaces here. An application usually has an *external interface* that it provides for users in order to offer the required functionality. An example would be the set of valid HTTP requests and responses provided by a web server. Aside from the external interfaces, applications also usually expose other details to the outsiders such as potential attackers, often as a result of the way they are implemented. These *internal interfaces* are not expected to be consumed by the outside world.

Malware, however, often depends on internal interfaces. This is due to the fact that if a malicious program only utilizes an external interface – the functionality an application is originally meant to provide – its capabilities would be severely restricted as it could only employ the functionality designed for an ordinary user. Malware therefore uses internal interfaces accidentally exposed by applications. We therefore concentrate on diversification of internal interfaces.

III. Proposed Targets of Diversification

To determine the feasibility of diversification on different IoT operating systems we looked at interfaces that are good candidates for diversification in ten different operating systems. This section discusses the interfaces and presents our findings.

A. Diversifiable Interfaces

Several different interfaces have been suggested as potential targets for diversification in the existing literature. Note that we use the term *interface* very broadly here. We do not only mean interfaces provided by software modules, but also, for example the command set of a language (like Bash) or memory addresses are considered as interfaces that are candidates for diversification. In other words, an interface is anything that is used to get access to essential resources of a device. The following interfaces were chosen as targets of our study:

1) *System calls.* A system call is the way programs use to request a service from the operating system and access the computer's critical resources [30]. In order to prevent the attacker from using system calls, we can uniquely change the system call numbers [22], [20], [25]. This diversification is then propagated to trusted binaries and libraries, which are diversified accordingly so that they are compatible with the diversified system and can call new secret system call numbers.
2) *Functions in shared libraries.* System calls can also be invoked indirectly by calling several wrapper functions in many operating system libraries. Malware should be prevented from accessing any critical resources by diversifying these entry points. To this end, any library functions that directly or indirectly issue system calls are diversified [12], [21]. Diversification is also propagated to libraries and applications that call these library functions.
3) *Command shell.* Malware does not always use system calls or library functions. Instead, it can make use of interpreted languages such as shell scripts, to achieve its goals. In a similar fashion as library functions, the language interface of the shell is also an entry point to the resources of a device. Many attacks such as ShellShock [14] have seized this opportunity. If we change the language interface – the set of tokens used by the command line interpreter – the attacks based on this known language interface will fail [32], [31], [24]. Shell scripts in the system are then diversified to correspond to the new secret language.
4) *Memory space.* Memory space can also be seen as a diversifiable interface. Address space layout randomization (ASLR) is a security approach that provides protection from buffer overflow attacks. In ASLR, the address space positions of the most essential parts of a process are randomly rearranged in order to make it more difficult for the adversary to jump to a specific position in the memory (such as an exploited function) [35]. The chance of an attacker guessing the location of certain randomly placed memory area is increased by upping the amount of virtual memory space where the diversification occurs. To achieve this, however, a memory management unit is needed. A memory management unit (MMU) is a unit through which the memory references are passed. It takes care of translating the virtual memory addresses to physical addresses.
5) *Protocols.* As IoT devices operate in a network, they need to use protocols to communicate. These protocols can also be seen as targets for diversification [17]. For example, the Constrained Application Protocol (CoAP)

TABLE I
DIVERSIFIABLE INTERFACES IN IOT OPERATING SYSTEMS

OS	Source	MMU support	System calls	Shell
GNU/Linux	Available	Required	Yes	Yes
Brillo	Available	Yes	Yes [9]	Yes
Contiki	Available	No	No [1]	Yes [18]
Integrity	Closed	Yes	Yes [3]	No
Neutrino	Closed	Yes [7]	Yes [8]	Yes
NuttX	Available	Partial	Yes [6]	Yes
mbed	Available	No / uVisor	Partial [4]	Yes
RIOT	Available	No	No [11]	Yes
TinyOS	Available	No	No [10]	Yes
VxWorks	Closed	Yes	Yes [26]	Yes

is a software protocol that enables simple electronics devices to communicate with each other [29]. It is an application layer protocol specifically designed for resource-constrained devices such as "things" in IoT. By diversifying a protocol, we can create a large number of uniquely diversified protocols from the original protocol. Thinking of a protocol as a state machine, this means that diversification can add new states and transitions to the original. Of course, this protection comes at the price of complication and some slowdown.

The first four diversification schemes have been suggested for traditional operating systems whereas the fifth is more IoT specific. In the context of IoT, things may be somewhat different. For example, some IoT operating systems do not provide an implementation of a system call interface or this interface is not widely used by applications. Also, not all IoT operating systems include a MMU. We will discuss these interfaces in the context of IoT next.

B. Interfaces Found in IoT Operating Systems

We studied the previously discussed interfaces in ten IoT operating system. Our findings are listed in Table I. We included many notable IoT friendly operating systems in the study, some commercial and many open-source. There certainly are more operating systems and new ones may be in the making as new use cases for IoT are found.

Table I shows the availability of MMU support, system call interface and shell for each studied operating system. Availability of source code is also listed as open-source systems are generally better suited for diversification, although diversification techniques can and often are also used on binary level [23].

In some cases, for example in the cases of RIOT and TinyOS, the number of interfaces to which we can apply diversification appears to be limited without concrete use cases. However, many operating systems like Linux and Brillo contain several essential interfaces fit for diversification. It is also an interesting observation that all the proprietary systems we studied seem to contain many diversifiable interfaces.

Brillo, an operating system geared towards IoT by Google is still in its infancy, but from a security and diversification perspective it looks promising. This is achieved by requiring hardware more rich in features [34] compared to most other devices used in the IoT domain. Since having a MMU has been rather scarce in the IoT environment, many other IoT operating systems have not even considered the extra protection afforded by potentially compatible devices that would support MMU.

In contrast to Brillo, the open-source operating system *Contiki* requires little memory, but also has no support for memory management units and therefore has no support for a system call interface. Diversifiable interfaces are challenging to study on an operating system as small as Contiki as there is not a whole lot more within the system than the kernel itself. Contiki does include a shell where the language of the shell could be diversified, but the shell is not required by any part of a normal system and is therefore optional. Another potentially diversifiable feature on Contiki is the networking protocol 6loWPAN [28], where the protocol could be regarded as an interface. Although 6loWPAN can be used with encryption [27], diversfication can be used in combination with encryption to provide additional security.

ARM's *mbed* operating system can include a supervisor to isolate system components from each other without a full MMU in the hardware [5]. This isolation includes supervisor calls that are mostly analogous to system calls and is therefore an interesting subject of study concerning use of diversification as an additional security reinforcement layer. The supervisor allows different isolated components to communicate with each other, which can be regarded as an internal interface that then could be attacked. This interface could be diversified in order to make it impossible for injected code to even be able to reach the supervisor functions.

The proprietary operating systems listed – Integrity, Neutrino and VxWorks – claim to support MMU and appear to have possibility for system calls. These calls could probably be diversified for similar additional security as in ARM's mbed depending on how they are implemented. Licensing may potentially limit the diversification of commercial systems due to restrictions on modifications, but this is out of the scope of this paper.

Most of the studied operating systems include a shell of at least some type. Several vulnerabilities such as Imagemagick CVE-20163714 [2] are based on unescaped input, which makes it appealing to study the possibility of diversifying the shell language. However, many IoT operating systems currently do not expose the shell for use to the rest of the system and the command shell is mainly geared towards debugging.

Although not covered in this study in detail, on a more fine-grained level there are also some operating system specific APIs that could diversified. For example, Contiki has a device driver API and a file system API. As these operating systems mature, most of them will probably contain more diversifiable interfaces, such as shared libraries and operating system APIs aimed to make building applications easier. Of course, the interfaces and libraries available for diversification also strongly depend on the application area and specific use case

of a particular device.

Finally, even though the software interfaces of the IoT operating systems we studied may not be perfect for diversification, they all support some diversifiable protocols. For example, Contiki offers a full IP stack which includes standard protocols like UDP, TCP, and HTTP [18]. Also, most IoT operating systems also support new low-power standards such as CoAP and RPL. All of these protocols can be seen as targets for diversification.

IV. Benefits and Limitations of Diversification

A. Benefits

Diversification has several benefits in the context of IoT operating systems:

- *Mitigate the threat of large-scale attacks*: As we have seen, billions of devices are currently distributed with identical, well-known interfaces. This software monoculture allows the adversary to build one exploit to attack all instances of the software. Diversification, by breaking this monoculture, is a promising proactive security mechanism for the widely distributed IoT operating systems.
- *Negligible overhead*: Resources of IoT devices are extremely limited, for example in terms of computational power and memory. Diversification usually has very small overheads or no overhead at all. For example, simply replacing the system call mapping with a new one has no performance penalties and no adverse effect on memory consumption. On the other hand, many other protection mechanisms like resource intensive anti-virus programs are not such a good fit for IoT devices.
- *Energy efficiency*: The previous point directly leads to the fact that diversification is also energy-efficient. With the huge number of IoT devices, conserving energy is one of the greatest challenges in IoT environment. Because its modest resource requirements, diversification is an ideal mechanism for providing a trade-off between the goals of good security and low energy consumption.
- *Alleviate the problem of irregular or absent updates*: One downside of embedded devices is that they often cannot be updated. If a vulnerability is found, the devices are exposed to malware. Diversification is an approach that does not remove the software vulnerabilities, but makes it difficult for the adversary to use them. Diversification is therefore proactive in the sense that it also provides protection for attacks that are currently unknown.
- *Orthogonality*: Diversification provides additional security and can be used in combination with other security measures like encryption.
- *Mitigate the propagation of malware*: Applying security diversification to IoT devices stops malicious programs from moving from node to node on the network. The malware can no longer spread as effectively as before.

B. Limitations

It should be clear from above that diversification brings great benefits to IoT systems. However, this security approach also has some limitations.

While diversification protects the software from malware by making its internal structure unexpected and hard to understand, some obfuscation techniques used for this purpose introduce costs with respect to execution time overhead, memory consumption, and increased size of programs.

The need to propagate diversification in a system also poses some challenges. For example, if system calls are diversified, all the applications and libraries using them have to be diversified accordingly. However, this can be done automatically with great accuracy, and only rarely requires help from a programmer [25].

Moreover, when diversification is applied to protocols, all devices involved have to support the diversified protocol. That is, the diversification in this context needs to be propagated outside one device and the diversification secret has to be shared. Still, we believe protocol diversification is applicable within small groups of hosts.

V. Conclusions and Future Work

In this paper, we have advocated applying diversification techniques on the operating systems and internal interfaces of the IoT devices to make it more difficult to successfully attack the devices. Also, we have suggested diversifying protocols used by these devices.

We have seen that because of the special characteristics of IoT and insufficient attention software security has received in this environment, security is more challenging compared to traditional computer systems. Because its low overhead and orthogonality, diversification of internal interfaces is a protection mechanism very well suited for the limited capacity and resources of IoT devices.

However, it seems that this method is not equally well applicable for all IoT operating systems, since some systems are lacking the interfaces that diversification is usually applied to in the existing literature. Still, we have seen that all operating systems have some interfaces that are good candidates for diversification, and many of them even seem to be fit for full-fledged multilayer diversification schemes. Most notably, diversification can be comprehensively applied on all devices using Linux, and Google's Brillo also looks very promising in this respect. Protocols like CoAP also provide opportunities for diversification.

For future work, it would be ideal to study how already existing IoT systems operate as a whole and determine interfaces worth diversifying. Operating systems possessing capabilities such as system calls and also, to some degree, of MMU support, seem to be a great target for diversification.

Also, a thorough study of the most common attack vectors used in exploits targeting IoT operating systems should be conducted. This would help in gaining more confidence on which operating systems and interfaces are good candidates for diversification.

The next step is to create a proof-of-concept implementation of a diversified IoT operating system. While there are certainly some challenges and differences to diversification of

traditional computer systems, we believe that diversification is a feasible and usable security measure for IoT operating systems.

ACKNOWLEDGMENT

The authors gratefully acknowledge Tekes – the Finnish Funding Agency for Innovation, DIGILE Oy and Cyber Trust research program for their support. This research is also supported by Tekes project 1881/31/2016 "Cybersecurity by Software Diversification".

REFERENCES

[1] Contiki Wiki. https://github.com/contiki-os/contiki/wiki. Accessed: 2016-06-23.

[2] CVE-2016-3714. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-3714. Accessed: 2016-06-20.

[3] INTEGRITY Real-time Operating System. http://www.ghs.com/products/rtos/integrity.html. Accessed: 2016-06-22.

[4] mbed OS uVisor. https://github.com/ARMmbed/uvisor. Accessed: 2016-06-22.

[5] mbed os uvisor. https://github.com/ARMmbed/uvisor. Accessed: 2016-06-22.

[6] Memory Configurations. http://nuttx.org/doku.php?id=wiki:nxinternal:memconfigs. Accessed: 2016-06-22.

[7] QNX Developer Support. http://www.qnx.com/developers/docs/6.3.0SP3/neutrino/sys$_arch/proc.html. Accessed : 2016 - 06 - 22$.

[8] QNX SDP 6.6 Documentation. http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.neutrino.getting_started%2Ftopic%2Fs1_procs_Starting_with_system.html. Accessed: 2016-06-22.

[9] SYSCALLS.TXT. https://android.googlesource.com/platform/bionic.git/+/brillo-m9-dev/libc/SYSCALLS.TXT. Accessed: 2016-06-22.

[10] TinyOS Documentation Wiki. http://tinyos.stanford.edu/tinyos-wiki. Accessed: 2016-06-22.

[11] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt. Riot os: Towards an os for the internet of things. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 79–80, April 2013.

[12] M. Chew and D. Song. Mitigating Buffer Overflows by Operating System Randomization. Technical report, Carnegie Mellon University, 2002.

[13] C. Collberg, C. Thomborson, and D. Low. A Taxonomy of Obfuscation Transformations. Technical Report 148, The University of Auckland, 1997.

[14] National Vulnerability Database. Vulnerability summary for cve-2014-6271. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6271. Initial CVE of Shellshock vulnerability. Accessed: 2016-06-23.

[15] Hewlett Packard Enterprise. Internet of things research study, 2015.

[16] Gartner. Gartner Says 6.4 Billion Connected things Will Be in Use in 2016, Up 30 Percent From 2015. http://www.vxdev.com/docs/vx55man/vxworks/guide/c-vm.html. Accessed: 2016-06-23.

[17] E. Hjelmvik and W. John. Breaking and improving protocol obfuscation. Chalmers University of Technology, Tech. Rep, vol. 123751, 2010.

[18] Contiki Homepage. http://www.contiki-os.org/. Accessed: 2016-06-23.

[19] S. Hosseinzadeh, S. Rauti, S. Hyrynsalmi, and V. Leppänen. Security in the internet of things through obfuscation and diversification. In *Computing, Communication and Security (ICCCS), 2015 International Conference on*, pages 1–5, 2015.

[20] X. Jiang, H. J. Wang, D. Xu, and Y. Wang. RandSys: Thwarting Code Injection Attacks with System Service Interface Randomization. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 209–218, Oct 2007.

[21] S. Lauren, P. Mäki, S. Rauti, S. Hosseinzadeh, S. Hyrynsalmi, and V. Leppänen. Symbol Diversification of Linux Binaries. In *Internet Security (WorldCIS), 2014 World Congress on*, pages 74–79. IEEE, 2014.

[22] Z. Liang, B. Liang, and L. Li. A System Call Randomization Based Method for Countering Code-Injection Attacks. *International Journal of Information Technology and Computer Science (IJITCS)*, 1(1):1, 2009.

[23] G. Portokalidis and A.D. Keromytis. Fast and practical instruction-set randomization for commodity systems. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 41–48, New York, NY, USA, 2010. ACM.

[24] G. Portokalidis and A.D. Keromytis. Global ISR: Toward a Comprehensive Defense Against Unauthorized Code Execution. In *Moving Target Defense, Creating Asymmetric Uncertainty for Cyber Threats, Advances in Information Security 54*, pages 469–480. Springer, 2014.

[25] S. Rauti, S. Laurén, S. Hosseinzadeh, J. Mäkelä, S. Hyrynsalmi, and V. Leppänen. Diversification of System Calls in Linux Binaries. In *The 6th International Conference on Trustworthy Systems (InTrust 2014)*. IEEE, 2014.

[26] W. River. Vxworks kernel programmer's guide, 2003.

[27] N. Sastry and D. Wagner. Security considerations for ieee 802.15. 4 networks. In *Proceedings of the 3rd ACM workshop on Wireless security*, pages 32–42. ACM, 2004.

[28] Z. Shelby and C. Bormann. *6LoWPAN: The wireless embedded Internet*, volume 43. John Wiley & Sons, 2011.

[29] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). Internet Engineering Task Force (IETF), 2014.

[30] A. Tanenbaum. *Modern Operating Systems*. Pearson, 2014.

[31] J. Uitto, S. Rauti, and V. Leppänen. Practical implications and requirements of diversifying interpreted languages. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference), year = 2016, pages Article No. 14 publisher = ACM*.

[32] J. Uitto, S. Rauti, J.-M. Mäkelä, and V. Leppänen. Preventing Malicious Attacks by Diversifying Linux Shell Commands. In *Proceedings of the 14th Symposium on Programming Languages and Software Tools (SPLST'15)*, CEUR Workshop Proceedings 1525, 2015.

[33] International Telecommunication Union. Overview of the Internet of things. Recommendation ITU-T Y.2060, 2012.

[34] A. Wee. Google brillio os-another os dedicated to internet of things. *Screen*, 2016.

[35] H. Xu and S. Chapin. Address-space layout randomization using code islands. *J. Comput. Secur.*, 17(3):331–362, August 2009.