Penetration Testing in the IoT Age

Chung-Kuan Chen, Zhi-Kai Zhang, Shan-Hsin Lee, and Shiuhpyng Shieh,
National Chiao Tung University

Internet of Things (IoT) objects offer new services but also pose new security threats. Due to the heterogeneity, large number, and resource constraints of these objects, new penetration testing tools and techniques are needed to complement defensive mechanisms.

Internet of Things (IoT) devices and services are now integral to most daily activities. However, the IoT brings not only added convenience but, by connecting more and more objects to the Internet, new security threats.[1] Many applications in IoT ecosystems, from smart homes to customized healthcare, contain sensitive personal information that can become the targets of network attacks.

Unfortunately, ensuring the security of IoT objects is not straightforward for three major reasons. First, the IoT's heterogeneous nature makes it vulnerable to many kinds of attacks. Second, heavyweight protection mechanisms are infeasible for resource-constrained IoT devices. Third, many IoT objects are deployed only once and thereafter are rarely maintained or updated.

PENETRATION TESTING

Due to these challenges, penetration testing (PT), which employs offensive attack techniques to discover vulnerabilities, is often used to complement defensive security methods before IoT objects are deployed. Because malicious attacks need only a single exploit to be successful, improving PT coverage is crucial. To enhance manual PT, security researchers use automated tools to carry out three types of specialized PT: interface testing, transportation testing, and system testing.

Interface testing targets interfaces that interact with external users or devices. Major vulnerabilities can exist in an application if its input validation mechanisms are not in effect. In the Open Web Application Security Project (OWASP) tester guidelines for IoT applications (www.owasp.org/index.php/IoT_Testing_Guides), the categories "insecure web interface" and "insecure network services," among others, would be addressed by interface testing.

Transportation testing focuses on misuse issues and design flaws in communication protocols and weak cryptographic schemes. In the OWASP guidelines, "insufficient authentication/authorization," "lack of transport encryption/integrity verification," and "privacy concerns" fall into this type of testing.

System testing examines firmware, OSs, and system services for implementation flaws, insecure system settings,

 Published by the IEEE Computer Society

and other known vulnerabilities. In the OWASP guidelines, "insufficient security configurability" and "insecure software/firmware" are relevant for system testing.

To cope with the heterogeneity and large quantity of IoT objects, we propose modularization of test modules to scale up all three types of testing. At the same time, due to IoT devices' resource limitations, intelligent approaches are desirable for generating test plans based on available test modules to reduce wasted resources and redundant effort while extending test coverage.

INTERFACE TESTING

Many user-facing IoT objects have web-based interfaces, and these can have various vulnerabilities. Among the most common is input validation failure. Unlike traditional web interfaces, which are linked to operations closely coupled with data manipulation, IoT object interfaces can also be linked to code-oriented operations such as controlling system programs. Code-oriented attacks such as command injection and code injection could be even more severe than data-oriented attacks. Improving input validation testing is thus critical for the IoT. Although testing web-based interfaces is our focus, the same modularization and intelligence mechanisms described below can be applied to other types of IoT applications.

Modularized design

Testers employ various techniques for different input validation vulnerabilities. However, these methods are conceptually similar in that they all crawl to the entry points and submit the test payload. Modularizing interface testing would make it easier to create testing tools for specific vulnerabilities and install them on demand. Moreover, algorithms could be developed in a more systematic way—for example, to implement adaptive, prioritized, or automutation test strategies.

Intelligent payload mutation

Because IoT objects can lack comprehensive input validation mechanisms, extending the coverage of test payloads is desirable. A widely used method, fuzz testing, employs randomly generated payloads, but this is inefficient due to resources wasted on meaningless inputs. An alternative is to exhaustively or randomly generate syntax-correct inputs. This method provides better test coverage but is still inefficient, as the space of syntax-correct inputs is usually large.

Modularizing interface testing would make it easier to create testing tools for specific vulnerabilities and install them on demand.

Intelligently mutating known payloads is a compromise between manual testing and exhaustive/random testing. Combining existing evasion techniques provides greater ability to circumvent validation mechanisms. In this case, conflicting or overlapping techniques should be manipulated carefully to prune unnecessary test cases.[2] On the other hand, converting payloads to syntactically or semantically equivalent payloads is worthy of further investigation. Syntactic mutation generates payloads with slight changes at the syntax level. For example, SQL code "`'or 1 = 1`" can be mutated to "`'|| 1 = 1`". Semantic mutation converts the whole payload to functional equivalent ones. For instance, "`id = 1 or 1`" is semantically equivalent to "`id = id xor 0`".

Intelligent entry-point crawling

Entry-point discovery in an IoT environment with numerous network services is difficult and time-consuming. Because service entry points can be dynamically generated, the links between them can be complex, and loops might be produced across IoT objects. In addition, a dispatcher might be built into an IoT application to manage entry points. As the dispatcher can be in either a centralized or distributed structure, a crawler should be able to discover as many entry points as possible in both types of structures to locate more test targets. A proof-of-concept vulnerability scanner that does this, VulScan,[2] has been developed to complement manual PT.

TRANSPORTATION TESTING

Transportation testing is performed both on the network infrastructure interconnecting IoT objects as well as the associated cryptographic schemes and communication protocols used to protect messages.

New network infrastructures

Messages between IoT objects traverse heterogeneous networks such as TCP/IP, Zigbee, and 6LoWPAN. To allow more efficient object communication, new infrastructures such as FIA (www.nets-fia.net), HUB4NGI (www.hub4ngi.eu), and PNS[3] have been proposed. New PT tools are needed to test these infrastructures, the protocols, and the gateways or converters between the infrastructures and protocols. Because network heterogeneity is a key issue in IoT communication, transportation testing should be modularized to provide better flexibility.

Cryptographic issues

In general, the cryptographic algorithms that protect network communication are believed to be secure due to theoretical proofs. When vulnerabilities are discovered, they're generally attributable to misuse, implementation failures, and bad protocol design. However, resource-constrained IoT objects can't afford heavyweight cryptographic mechanisms. Moreover, messages between devices usually are well formatted and lack entropy. The combination of these factors could make differential cryptanalysis or statistical attacks possible.

Trusted platform modules (TPMs) enable new applications but also raise new threats. For example, the ROCA vulnerability[4] is caused by a weak prime-number generator in the RSA library within TPMs. This vulnerability affects many vendors including Microsoft, Google, and HP. Another example is KRACK attacks,[5] which exploit a flaw in Wi-Fi's WPA2 encryption and affects all major software platforms. As cryptographic operations are rarely computed in cleartext, developing PT methods to discover such vulnerabilities in the IoT is challenging.

SYSTEM TESTING

In contrast to interface testing, which focuses on commonly used technologies such as web interfaces, proprietary programs are the main targets of system testing. Without having knowledge of such systems, testers often resort to black-box methods, such as fuzz testing. Given the large number of IoT objects to be tested, exhausting all test cases is infeasible. It's therefore helpful generating test cases through automatic reverse-engineering, what is termed grey-box PT.

In conventional computing environments the x86/x64 instruction-set architecture (ISA) dominates, but other ISAs such as ARM, MIPS, and PPC are also used in the IoT. OSs vary among IoT objects as well, with general-purpose OSs such as Linux, Windows, and Android often customized. The diversity of IoT objects makes automated reverse-engineering challenging.

Encapsulation

To mitigate the impacts of system diversity, encapsulation can enable cross-platform analysis. Encapsulation involves using an abstract language such as LLVM (http://llvm.org) or VEX (http://valgrind.org) to create an intermediate representation (IR) of different machine languages to emulate ISAs. Hardware-assisted emulators are used to test programs running on specific ISAs, but software-based emulators such as QEMU (www.qemu.org) and Bochs (http://bochs.sourceforge.net) can leverage multiple ISAs and are more suitable for IoT objects. Another method for building an IR is symbolic execution, which translates a program to mathematical constraints and evaluates whether certain properties can be satisfied. With these constraints, developing an intelligent PT method with a more formalized foundation is possible.

To mitigate the impacts of system diversity, encapsulation can enable cross-platform analysis.

Virtual machine introspection

While symbolic execution mostly deals with per-process information, system-wide runtime information is also important for PT. However, runtime analysis tools might not be available for IoT objects. Due to resource constraints, object diversity, and proprietary architectures, developing debugging and analysis tools for different objects is usually infeasible. An alternative approach is virtual machine introspection (VMI), which monitors VM execution in the hypervisor outside the VM.[6,7] Because VMI doesn't modify the guest OS, IoT objects are easier to deploy. Through VMI, the emulator's out-of-box monitoring, memory forensics, and debugging features can be developed more easily to enable both manual and automatic PT.

Intelligent grey-box testing

As the boundary of grey-box testing is more obscure than white- and black-box testing, a systematic division of testing phases enables the development of future testing techniques. Intelligent grey-box PT can be divided into four phases: vulnerability model construction, execution path exploration, vulnerability path searching, and vulnerability path verification. To discover vulnerabilities, the model of abnormal behaviors is first constructed. Next, control flows are analyzed to find each execution path. The vulnerability risk for each path is then estimated using information from the IR and VMI to prioritize testing order. Once the path with highest risk is identified, the symbolic execution resolves inputs to the path. During the final phase, if the resolved input is available, a verifier can monitor the program with the input to check whether the vulnerability model can be satisfied. Using this systematic approach, intelligent grey-box PT can discover system-level vulnerabilities.

To cope with the heterogeneity, large number, and resource constraints of IoT objects, PT tools and techniques should apply the principles of modularization and intelligence. Modularization provides the flexibility to test various targets, and intelligence enlarges test coverage and improves accuracy. In interface testing, input validation mechanisms should be tested using an intelligent mutation engine and entry-point

discovery automated. Transportation testing must address the problem of messages between IoT objects traversing heterogeneous networks. To deal with emerging IoT network infrastructures, PT tools should be compatible with the overlay networks. Cryptographic misuse issues and implementation flaws must also be considered. In system testing, the challenge is IoT objects with various ISAs and OSs. If encapsulation and related translation modules are available, cross-platform analysis becomes feasible. VMI and symbolic execution can be applied on top of encapsulation. In this way, intelligent analysis methods can be used to discover vulnerabilities in variant platforms.

REFERENCES

1. Z.-K. Zhang et al., "IoT Security: Ongoing Challenges and Research Opportunities," *Proc. IEEE 7th Int'l Conf. Service-Oriented Computing and Applications* (SOCA 14), 2014, pp. 230–234.
2. H.-C. Huang et al., "Web Application Security: Threats, Countermeasures, and Pitfalls," *Computer*, vol. 50, no. 6, 2017, pp. 81–85.
3. Z.-K. Zhang et al., "Identifying and Authenticating IoT Objects in a Natural Context," *Computer*, vol. 48, no. 8, 2015, pp. 81–83.
4. M. Nemec et al., "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli," *Proc. 2017 ACM SIGSAC Conf. Computer and Communications Security* (CCS 17), 2017, pp. 1631–1648.
5. M. Vanhoef and F. Piessens, "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2," *Proc. 2017 ACM SIGSAC Conf. Computer and Communications Security* (CCS 17), 2017, pp. 1313–1328.
6. K. Nance, M. Bishop, and B. Hay, "Virtual Machine Introspection: Observation or Interference?," *IEEE Security & Privacy*, vol. 6, no. 5, 2008, pp. 32–37.
7. C.-W. Wang et al., "Cloudebug: A Programmable Online Malware Testbed," *Computer*, vol. 47, no. 7, 2014, pp. 90–92.

CHUNG-KUAN CHEN is a PhD candidate in the Department of Computer Science at National Chiao Tung University (NCTU). Contact him at ckchen@cs.nctu.edu.tw.

ZHI-KAI ZHANG is a PhD candidate in the Department of Computer Science at NCTU. Contact him at skyzhang.cs99g@g2.nctu.edu.tw.

SHAN-HSIN LEE is a PhD student in the Department of Computer Science at NCTU. Contact him at shlee.cs06g@nctu.edu.tw.

SHIUHPYNG WINSTON SHIEH is a university chair professor and past chair of the Department of Computer Science at NCTU. Contact him at ssp@cs.nctu.edu.tw.

This article originally appeared in Computer, *vol. 51, no. 4, 2018.*