3.4 Rikke Kuipers, Ari Takanen/ Fuzzing embedded devices

3.4.1 Rikke Kuipers

Rikke is a security specialist, has a network engineering background at several ISPs and the public broadcasting platform in the Netherlands. After his move to Northen Finland he started working at Codenomicon, performing audits and security research. Besides his main focus on fuzzing (network)protocols he has a huge interest in web application hacking and information security in general. Ongoing projects are the development of an automated web auditing framework and tools, research on DVB fuzzing and writing whitepapers on various topics.

twitter: @rikkekuipers

3.4.2 Ari Takanen

Ari Takanen is the founder and CTO of Codenomicon, has been active in the field of software security research since 1998 focusing on information security issues in next-generation networks and security critical environments. In his work he aims to ensure that new technologies gain public trust by providing means of measuring and solidifying the quality of networked software. Ari Takanen is one of the people behind the PROTOS research project, which studied information security and reliability errors in e.g. WAP, SNMP, LDAP, VoIP implementations. Ari is the author of several papers on security, and is a frequent speaker at security and testing conferences, leading universities and international corporations. He is also the author of two books on VoIP security and security testing.

twitter: @aritakanen

3.4.3 Fuzzing embedded devices

This talk focuses on the increased connectivity of electronic devices commonly found in offices and homes. These devices are converging to a point where they deliver similar base functionalities and services. This often means more specialized hardware and integration of network protocols to enable universal communication. Internet connectivity exposes these devices traditionally situated in a closed environment now to the whole world, meaning exposure to threats from internal and external networks. This requires a new look on the security of these devices. A device is now expected to perform more functions than just its original purpose. For example, for years phones were primarily used for voice communication using the GSM standard. A common smartphone these days can be thought of as hand-held computer integrated with a mobile telephone, capable of displaying movies, pictures, browsing the Internet and staying up-to date on social networks using a wide variety of protocols. Growth in demand for advanced mobile devices boasting powerful processors, abundant memory, larger screens, and open operating systems has overtaken the rest of the mobile phone market. The same shift has now moved to the TV world. Vendors are developing their own platforms for Internet-enabled TVs, designed to connect directly to the Web. These devices are capable of displaying dynamic content from the web, browsing the Internet and accessing social media through the numerous widgets available. Standard functions usually also include playback of movies, pictures and music from various sources such as USB sticks or SD-cards. All these inputs combined define the attack surface for the TV. Fuzzing enables us to stress all this freshly exposed attack surface to test its security and robustness. It is designed to simulate real-world hacking attempts against devices by creating and sending malformed and unexpected messages (anomalies) with the intention to disrupt services. Fuzzing finds vulnerabilities which can potentially be exploited. Malformed, anomalous input such as field overflows / underflows could expose vulnerabilities in software which can then be patched before deployment. The talk will demonstrate how to use fuzzers to find flaws in your software, using most recent TVs (2012 models). This in the hope of a wider adoption of fuzzing in general, and thus improving product quality and security.

- Talk and paper can be downloaded from `http://grehack.org`

Fuzzing embedded devices

Finding unknown vulnerabilities in electronics

Rikke Kuipers
Codenomicon Ltd.
Oulu, Finland
rikke.kuipers@codenomicon.com

Ari Takanen
Codenomicon Ltd.
Oulu, Finland
ari.takanen@codenomicon.com

I. Topic intoduction

The increased complexity of new technologies and faster software release cycles are making traditional reactive security solutions ineffective. Instead, more adaptable, preemptive product security testing is needed. Moreover, for example, due to agile development and outsourcing, the software development process itself is also becoming more complicated further increasing the need for more effective product security practices. Fuzzing is a black box testing technique which can be used to efficient and effectively test the open interfaces of software. Using these protocol fuzzers, security vulnerabilities can be found and eliminated proactively in an easy and uniform way.

Fuzzers and other techniques to test software security have been around for quite some time now, so it should be possible to assume that recent software implementation are secure.

This paper will take a closer look at fuzzing and why it should be done, recent changes in the electronic landscape as I see them happening, and we take a closer look at protocol implementations in one fast upcoming new technology; Internet-enabled TVs.

II. Vulnerabilities

Vulnerabilities are not created, when a system is being attacked; they enable attacks. Vulnerabilities are implementation errors that are introduced into the code, when the programmer makes a mistake. They become vulnerabilities, once the software is released exposing the software for attacks. Security researchers, security companies and hackers discover some of the vulnerabilities, and if they choose to report the findings, they can enable software developers to create patches for the found vulnerabilities. However, other vulnerabilities still exist in the code waiting to be exposed. These unknown vulnerabilities are called zero-day vulnerabilities. Software vendors are unaware of their existence, thus there are no ready patches for them. Once a zero-day vulnerability is triggered, the developers race against the clock to get it fixed, before irrevocable damage is done to the company's sales or reputation.

Attackers need to find vulnerabilities in a device or a system in order to devise an attack against it. Basically, any crash-level bug can be exploited to attack a system or an application. Attackers send unexpected inputs to a system, and if they can get an abnormal response from the system, they continue to refine their inputs until they get the system to behave the way they want. Sometimes bugs can be exposed by simple individual inputs, and sometimes attackers have to communicate with longer message sequences with the system, in order to gain access to the deeper protocol layers. In some cases, vulnerabilities can even be triggered by events like heavier than normal use or maintenance. Therefore, the best way to ensure the security of your systems is to build security into it by finding and fixing these critical bugs before they become security vulnerabilities. The goal of fuzzing is to produce better quality software by finding vulnerabilities in the code before deployment.

III. Fuzzing

Fuzzing[1][2] is a form of attack simulation, in which unexpected data is fed to the system through an open interface, and the behavior of the system is then monitored. If the system fails, for example, by crashing or by failing built in code assertions, then there is a bug in the software. Similar to advanced code analysis tools, more advanced fuzzing tools also test multiple layers. It is important that the fuzzers can genuinely interoperate with the tested system. Only this way can they access the deeper protocol layers and test the system more thoroughly. In contrast to static analysis, fuzzing can only find bugs, which can be accessed through an open interface. However, all the found issues are critical and exploitable. Exploitation can be in the form of crashes, denial of service, security exposures, performance degradation or anomalous behavior in general.

The problem with traditional software testing techniques such as static analysis is that they also reports bugs that the software does not contain, namely false positives. Because it is impossible to test the entire system, test tools have to make a number of approximations, which leads to inaccuracies. False positives are troublesome, because checking them consumes valuable resources. Thus, some tools have adopted heuristic or statistic approaches. The advantage of these approaches is that they can reduce the amount of false positives without compromising test coverage. In effect, it can improve a tools ability to catch critical vulnerabilities.

In fuzzing, there are no false positives: Every bug is discovered as a result of a simulated attack. Thus, they are all real security threats. Simple fuzzing techniques rely on random mutation to create test cases. The coverage of mutation based fuzzers is limited and thus they only find some of the vulnerabilities the software contains. Yet, this is better than performing no fuzzing at all, because these are the vulnerabilities the attackers would also find. More sophisticated fuzzing techniques improve coverage by using protocol specifications to target protocol areas most susceptible to vulnerabilities. This type of optimized test generation also reduces the amount of test cases needed without compromising test coverage.

IV. Changes in the electronic landscape

In recent years the landscape of consumer media electronics has been formed by a set of devices, each performing its own dedicated function. A DVD player plays DVDs, a digital picture frame displays digital photos, a computer handles various data and so on. As technology advances and matures, customers' demands change.

A specific device is now expected to perform more functions than just its intended purpose. For example, mobile phones were primarily used for voice communication using the GSM standard during the last 20 years. A common smart phone these days can be thought of as hand-held computer integrated with a mobile telephone, capable of displaying movies, pictures, browsing the Internet and staying up-to-date on social networks using a wide variety of protocols. Growth in demand for advanced mobile devices boasting powerful processors, abundant memory, larger screens, and open operating systems has outpaced the rest of the mobile phone market for several years.

V. TV evolution

The same shift has now moved to the TV world. Vendors are developing their own platforms for Internet-enabled TVs, designed to connect directly to the Web and display content such as YouTube videos, weather reports and streaming movies or television shows. Some of the more expensive models even have full-blown browser implementations. Social media can be easily accessed by using one of the numerous widgets available for download. Standard functions usually also include playback of movies, pictures and music from various sources such as USB sticks or SD-cards.

Most of these devices are very similar to a normal desktop computer: they have a processor, memory, a hard disk and some sort of OS running. With these similarities in hardware and software it's very likely that these Internet TVs will face the same security implications as those existing for computers nowadays, in addition to new attack vectors.

VI. Attack surface

The attack surface can be divided in active and passive. The active attack surface refers to all the attack vectors of a given device, where the attacker has to actively transmit data to perform the attack. For TV this is mostly determined by the various services running on TV. The Bluetooth and wifi stack[4] are more and more commonly found in TVs, and are definitely interesting and well-known attack vectors in other embedded devices.

The passive attack surface refers to interfaces the attackers can "listen in on" to gain information without altering the data in any way. The attack surface depends heavily on the type of TV.

Televisions can be categorized as following:

A. Dumb DVB-enabled TVs

The so-called dumb TVs do not have much in terms of connectivity, except for the DVB. The standard Digital Video Broadcasting (DVB) protocol is the successor of analog broadcasting used in more than 80 countries worldwide. There are many DVB standards, each developed for its own intended use. The first of the DVB standards to be agreed upon by ETSI and others was the DVB-S standard (1994) for satellite transmission. DVB-T is used for terrestrial transmissions, and was commercialized around 1997. DVB-C is used in cable transmissions.

B. Media center TVs

Media Center TVs provide basic network connectivity and external media support from USB connectors and memory card readers. Network connectivity may appear to be limited to the local area network, providing media services and firmware updates.

Besides UPnP and DLNA type media center protocols and a basic IP-stack, DHCP and HTTP/FTP connections are used just to fetch LAN based media content or firmware upgrades over WAN. These TVs may host some low level network services in themselves.

C. Internet-enabled TVs

In addition to the functionality of the Media Center TVs these even more advanced TVs host applications, applets and widgets which give a channel to pull dynamic content directly from the Internet, typically from various web services. New applications and application updates can be fetched and installed directly from the Internet.

This level of openness in terms of attack surface is very similar to current smart phones or tablets, both of which have a connection to the Internet. The TV's browser may be directly used to access a wide variety of external content, exposing it to an even wider variety of attacks.

VII. DVB: NEW ATTACK VECTOR

A very interesting protocol to test in TVs is DVB, due to its wide implementation in TVs and demultiplexers. The DVB protocol is not used just to transmit video/audio streams to TVs. It is also used for local communication (i.e. car to car transmission), navigational purposes, access to content on hand-held devices (DVB-H), and even to provide Internet access in remote locations where no cable or mobile communication is possible (IP-over-DVB/MPEG). Besides the commercial implementation the DVB-S2, second generation DVB satellite signaling, is a protocol commonly used by military instances all over the world to provide communication and tactical insight. Like most protocols, the DVB implementations in devices can be subject to vulnerabilities when not properly implemented.

On a high level, content providers such as cable companies deliver one DVB stream to the TV. This stream contains several channels, each on its own frequency. The channels are combined, or "multiplexed" into one stream and delivered to the TV, which "demultiplexes" the signal so it can be "read" from the various channels. In addition to the audio/video streams (the payload), there are also a number of tables included in the so-called transport stream. These tables provide the TV with information about the stream. An example is the "PAT", which stands for Program Association Table, listing all available programs in the transport stream.

Depending on the guidelines set per country, a TV can be picky about which tables should be present in the transport stream. When fuzzing the DVB, to mimic the exact stream a TV would normally accept, we used a capture device to record directly from the DVB-T/C source. This captured transport stream was then run through our fuzzer (Defensics MPEG TS suite) in order to create fuzzed version of the transport stream. In the last step we muxed the transport streams into a DVB signal and sent it back to the TV using a modulator.

VIII. INSTRUMENTATION

Normally when a protocol is fuzzed, it’s important to know when a test case triggers anomalous behavior in the target under test. This is done using instrumentation, using the following sequence:

A. Interoperability

Several valid sequences are taken from the RFC of the protocol. It is then verified that the subject under test (SUT) replies according to the specification when this sequence is tested. It could be that only some of the RFC specified sequences are implemented in the SUT. All of those that are found to interop, will be used in the fuzzing. This is only done once, before the actual fuzzing starts.

B. Anomalized packet

Based on packet definition in the RFC of the protocol, a packet is built with one or more anomalies. Anomalies could range from field level anomalies (integer anomalies, overflows in field, etc), structural level (underflows, repetition of elements, unexpected elements, etc) to sequence level anomalies (out of sequence, omitted, repeated packets, etc). The fuzzed packet is send to the SUT, which then might or might not reply. A reply could be in the form of an error response, a valid response, or an anomalous response.

C. Instrumentation round

One predefined sequence, proven to interop with the device in phase A is rerun against the SUT. If the device responds within a given timeframe and according to RFC specifications, the test case is marked as a pass. The next anomalized packet can now be send (phase B) again. If the SUT fails to respond in time, or responds with different packet than defined, the test case is marked as failed.

When we tested DVB there was no instrumentation possible by using just the protocol since it’s mostly one way traffic. To solve this problem we used an ICMP heartbeat to verify the health of the TV under test, using the network connectivity of the TV.

IX. Threat Scenarios

A. Denial Of Service attacks

Crashing the services in a TV so that a manual reboot of the TV is required for it to operate again. In some cases a firmware flash from the manufacturer is needed to fix the permanent damage caused by the attack.

B. Exploits

When a discovered vulnerability is analyzed, debugged and turned into a working exploit, the TVs are even more defenseless than computers. Malicious code can be run on the TV to gain unauthorized access to the TV. Counter-measurements such as memory protection technologies are not available in most TVs, making them easy targets. Detecting a breach can be hard because diagnostic tools for TVs are not available and users have no operating system level access to the TV.

C. Covert Malware

Botnets and other espionage software can be installed and remain undetected in home entertainment systems.

D. Loss of sensitive data

With the rise of "on-demand" services, such as the ability to rent movies and watch TV shows within minutes, the TV has to support some payment options. Credit card numbers are saved on the TV itself, which implies a possibility that could be extracted. Services like email and access to social media that are available on the more advanced TVs pull and store sensitive data on the TV, which could be accessed by an attacker.

E. Social Engineering and Static Media

A much underestimated vector is the delivery of content to a TV by static media such as USB sticks and memory cards. The software on a TV decodes movie files and displays pictures, which can be malicious and cause crashes or buffer overflows. Users can be tricked to inserting malicious USB media into the TV.

X. Results from Smart-TV fuzzing

Protocol / TV	TV 1	TV 2	TV 3	TV 4	TV 5	TV 6
IPv4	pass	fail	fail	pass	pass	fail
DVB	fail	fail	fail	fail	fail	fail
UPnP	n/a	fail	pass	n/a	n/a	fail
Images	pass	fail	fail	n/a	n/a	fail
Audio	pass	pass	n/a	n/a	n/a	pass
Video	fail	fail	n/a	fail	fail	fail

Table I: Protocols tested per TV

While testing the IPv4 stack we discovered vulnerabilities in some of the TVs. Unfortunately none of the TVs tested were IPv6 capable. As can be seen from the result table, all of the TVs crashed with several protocols. Surprisingly, protocols which have been around for a very long time, such as IPv4, still cause some models to crash or malfunction. This could be due to the use of an old kernel, a custom stack implementation or a CPU with insufficient power to handle larger or fragmented packets. [5]

Most failures occurred in what should be the TV's primary function: decoding multimedia. Simple fuzzed images crash the TV with ease, and consequently they present an interesting attack vector due to the numerous way of distributing the possible exploits to the victim such as embedding the hostile images into web pages.

Video decoding generated crashes on most TVs as well. These crashes were found using the same video sample on each of the TVs. This means that not all of the codecs within the TVs were tested, when one simple one caused such crashes.

The DVB protocol seems to be vulnerable throughout all of the tested smart TVs, which is disappointing when considering that it is one of the key features a TV is supposed to handle.

The DVB is an interesting attack vector for several reasons. In the case of "simple" customer devices such as TVs, these kinds of vulnerabilities could be triggered remotely by overpowering the DVB-T signal in the air. Another possible attack would be to buy airtime from the commercial break, provide the broadcaster with your normal looking shampoo commercial, i.e. your prepared transport stream, that would contain exploits for the top 10 Internet-enabled TVs. Where our research was limited by availability of possible test targets, in real life that is not an issue. DVB implementations in any device could be remotely crashed, get infected with malware, be eavesdropped upon, or at worst, get controlled by the attackers.[3]

XI. Conclusion

The solution is to test for security. Black-box testing techniques like fuzzing should be integrated in the software development lifecycle to eradicate security vulnerabilities before rolling the TVs out and exposing consumers to the threats. As shown in this study, the TV-specific protocols are likely to open up new attack vectors, and those all can be used to attack the TV set. Zero-day vulnerabilities in communication protocols open the devices to remote compromise. The TV manufacturers should be able to provide updates in a quick and easy manner for their deployed TVs. Open source software used in TVs poses a threat of its own, as known vulnerabilities in open source are always public. Reacting to found open source bugs has to be immediate, which is not always possible with TVs. It could be that TV need to be rebooted before patches are applied, or that the TV needs to be idle, i.e. on standby. It could also be that TVs have no internet access, but only local connectivity. Automated patch release process has to be in place if security bugs are not found proactively, before release. Which could be costly to develop and maintain.

REFERENCES

[1] Ari Takanen, Jared DeMott and Charlie Miller, “Fuzzing for Software Security Testing and Quality Assurance (Artech House Information Security and Privacy)”, 30 June 2008.

[2] Rauli Kaksonen, “A Functional Method for Assessing Protocol Implementation Security”, Technical Research Centre of Finland, VTT Publications, 2001.

[3] Rikke Kuipers, Eeva Starck and Hannu Heikkinen, “Smart TV Hacking: Crashing Your Home Entertainment”, Codenomicon Ltd, Codenomicon technical report, 2012.

[4] Tommi Mäkilä and Jukka Taimisto, “Intelligent Bluetooth Fuzzing - Why Bother?”, Codenomicon Ltd, Codenomicon technical report, 2011.

[5] “Microsoft Security Bulletin MS11-083 - Critical Vulnerability in TCP/IP Could Allow Remote Code Execution (2588516)”, http://technet.microsoft.com/en-us/security/bulletin/ms11-083, 2011