Duqu: Analysis, Detection, and Lessons Learned

Boldizsár Bencsáth
CrySyS Lab, BME
boldi@crysys.hu

Gábor Pék
CrySyS Lab, BME
pek@crysys.hu

Levente Buttyán
CrySyS Lab, BME
buttyan@crysys.hu

Márk Félegyházi
CrySyS Lab, BME
mfelegyhazi@crysys.hu

ABSTRACT

In September 2011, a European company sought our help to investigate a security incident that happened in their IT system. During the investigation, we discovered a new malware that was unknown to all mainstream anti-virus products, however, it showed striking similarities to the infamous Stuxnet worm. We named the new malware Duqu, and we carried out its first analysis. Our findings led to the hypothesis that Duqu was probably created by the same people who developed Stuxnet, but with a different purpose: unlike Stuxnet whose mission was to attack industrial equipment, Duqu is an information stealer rootkit. Nevertheless, both pieces of malware have a modular structure, and they can be re-configured remotely from a Command and Control server to include virtually any kind of functionality. In this paper, we present an abridged version of our initial Duqu analysis, which is available in a longer format as a technical report. We also describe the Duqu detector toolkit, a set of heuristic tools that we developed to detect Duqu and its variants. Finally, we discuss a number of issues that we learned, observed, or identified during our Duqu analysis project concerning the problems of preventing, detecting, and handling targeted malware attacks; we believe that solving these issues represents a great challenge to the system security community.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and protection—*invasive software*

General Terms

Design, Experimentation, Security

Keywords

targeted attacks, advanced persistent threat, malware analysis, malware detection, incident handling, forensics, Duqu, Stuxnet, Tilded platform

EuroSec'12, April 10, 2012, Bern, Switzerland.

1. INTRODUCTION

In September 2011, a company from Europe approached us to help them in the investigation of a security incident that occurred in their IT system. The NDA that we signed with the company does not allow us to reveal more information about the company itself and the details of the incident. However, the company allowed us to share information about the root cause behind the incident, which happens to be a malware that was previously unknown. When we discovered this malware during the investigation of the incident, we gave it the name Duqu, because it has an infostealer component that creates files in the infected system with filenames starting with the string "~DQ".

Since our discovery, Duqu has become widely known and was covered by the media, mainly due to its striking similarity to the infamous Stuxnet worm in terms of design philosophy, internal structure and mechanisms, implementation details, and the estimated amount of effort needed to create it. However, no paper has discussed this malware in the scientific community so far. Yet, we believe it is important to discuss Duqu within the scientific community too, because the success of malware codes like Duqu and Stuxnet is due to the inefficiency of our current defense mechanisms. Designing efficient security mechanisms against targeted attacks requires a joint effort of researchers, industry and policy-makers.

Our main contribution related to Duqu is threefold:

1. *Discovery:* First of all, we discovered and named Duqu, and we performed its first analysis. The main outcome of our analysis was that Duqu is extremely similar to the Stuxnet worm in its design and implementation, but there are also obvious differences between them stemming from their different objectives. These findings have later been confirmed by others, and led many to believe that Duqu was probably created by the same people who developed Stuxnet, but with a different purpose: unlike Stuxnet that infected PLCs and maliciously controlled uranium centrifuges, Duqu is an information stealer rootkit targeting MS Windows based PCs. We dumped our analysis results in a confidential report that we shared only with a small set of mainstream anti-virus vendors and security experts. We also shared with them the Duqu samples that we had, such that they can repeat and extend our analysis. In a very short amount of time, Symantec confirmed our findings, extended our analysis, and published the first public Duqu report [4] on October 18, 2011. A reduced and anonymized version of our

initial analysis appeared in the Symantec report as an appendix. A few days later our lab has been identified as the source of the anonymized appendix[1] based on some cryptographic hash values computed from some Duqu components and placed on the personal blog site of one of us for monitoring purposes[2].

2. *Dropper:* Once the Duqu samples have been shared among the anti-virus vendors, they updated their products such that they could detect Duqu. This was an important step, but a key element was still missing: no one knew how Duqu infects the first computer in a network. Our second main contribution was that we identified the dropper of Duqu, which was an MS Word document with a zero-day kernel exploit in it. To prove that it is a zero-day exploit, we opened the dropper file on a fully patched system, and observed how Duqu installs itself. However, the difficulty was that the installation does not start immediately, only if the computer is idle for 10 minutes and some other conditions hold. It took us some time to find all these conditions. We immediately notified Symantec and Microsoft about our findings including the conditions for successful installation. We also helped Symantec to reproduce the installation of Duqu from the dropper in their analysis environment, such that they can confirm our results. Symantec then produced an anonymized dropper file with a proof-of-concept exploit code, and that was shared with Microsoft and others such that they can take the necessary steps for fixing the problem. In effect, the exploit took advantage of an unknown bug in the handling of embedded fonts in the Windows kernel; this bug was fixed by Microsoft in December 2011.

3. *Detection:* After the analysis it was clear to us that Duqu generated anomalies in the infected systems that could have been rather easy to spot. Yet, Duqu was not detected by any anti-virus product at the time. Based on the lessons learnt, we developed a Duqu detector toolkit and made it available under an open source license for free[3]. Our toolkit consists of simple heuristic tools, which are individual programs that can be run on a system to look for a certain type of anomaly, such as PNF files without corresponding INF files, and drivers with too large entropy (which suggests that the file is obfuscated). The open source license allows users to check the precise operation of the detector and to create their own executables with their trusted compilers. This allows for the usage of our Duqu detector toolkit in critical infrastructures, where commercial anti-virus products may not be used due to lack of trust in their vendors. As a heuristic tool, our detector may generate false positive alarms, but we believe that in critical infrastructures, it is affordable to invest some time in filtering false positives, and this additional effort is preferred to missing a real attack. The positive side is that our heuristic tools may detect as yet unknown variants of Duqu, or even Stuxnet, and they may also detect remains of a past infection. Our Duqu detector has been downloaded from more than 12000 distinct IP addresses from all around the world.

In this paper, we give a brief overview of our first Duqu analysis results, focusing on the evidence that we found on the similarity between Duqu and Stuxnet. More details can be found in our full technical report [1]. In addition, we describe our Duqu detector toolkit. Finally, we discuss some issues that we observed or identified during the unfolding of the Duqu story concerning the problems of preventing, detecting, and handling targeted malware attacks. We hope that these issues will generate discussion within the system security research community, and open new research directions.

2. DUQU ANALYSIS

In order to investigate the reason of the incident at the company that requested our help, and to be able to fix the system such that the same type of incident cannot happen again, we were allowed to access the hard drive of an infected computer. We produced a virtualized copy of the affected computer, which allowed us to revert the machine to a former state at any point during the analysis.

We hypothesized from the beginning that the malware loads a kernel driver, so the first task was to find that driver. The driver called `cmi4432.sys` became suspicious, but it was inactive on the particular computer we investigated, and it was digitally signed. Therefore, we performed a systematic search: we deleted groups of kernel drivers until we found that the malware was no more active, then refined the search iteratively to pinpoint the driver that was part of the malware. This led to the identification of another driver called `jminet7.sys`. After that, we recognized that `cmi4432.sys` is indeed connected to the threat, and we have already known that `cmi4432.sys` was related to some suspicious PNF files that did not have any corresponding INF files installed on the system.

Finally, we uncovered three main groups of malware components: a standalone keylogger, a group of objects related to `jminet7.sys`, and another group of objects related to `cmi4432.sys`. The keylogger is a standalone executable that was found on an infected computer. It contains an internal encrypted DLL, which provides the keylogging functions, whereas the main executable injects the DLL and controls the logging process. The jminet7 group of objects work as follows: In the registry, a service is defined that loads the `jminet7.sys` driver during the Windows bootup process. This kernel driver then loads configuration data from itself and from the registry, and injects code from a DLL called `netp191.pnf` into a system process. Finally, some configuration data is stored in an encrypted configuration file called `netp192.pnf`. The cmi4432 group of objects essentially exhibit the same kind of behavior, but they use the files `cmi4432.pnf` and `cmi4464.pnf`.

This sort of behavior was very similar to the operation of Stuxnet. Our suspicion that Duqu and Stuxnet are related grew rapidly when we discovered that Duqu also injects code into the `lsass.exe` process, it uses non-existent virtual files, and it uses the same hooks from `ntdll.dll` as Stuxnet. In addition, as we said before, the driver `cmi4432.sys` had a

[1] `http://www.symantec.com/connect/blogs/duqu-status-update-1`

[2] We placed these hashes on the blog site to see if there is anybody looking for them.

[3] `http://www.crysys.hu/duqudetector.html`

valid digital signature on it. The corresponding certificate belonged to a Taiwanese company that did not seem to be the author of the driver, so we suspected that the signature was generated with a compromised private key. The only known case at that time where malicious kernel drivers were signed with possibly compromised keys was the case of Stuxnet, and the compromised keys belonged to Taiwanese firms in that case too.

Given the strong evidence for a highly sophisticated malware, we decided to carry out a deeper analysis of Duqu, including mainly static binary analysis, trying to see if it is really related to Stuxnet. The potential connection between the two incidents urged us to reveal Duqu's existence to the security community as soon as possible. Thus, we set 10 days as a hard deadline for ourselves to finish the analysis; we did not aim at completeness, rather we wanted to understand as much as possible within 10 days and then release our analysis report. Below, we summarize the key findings of our analysis. Due to space limitations, we keep the discussion brief, and we refer the interested reader for more details to our technical report [1].

Decryption keys and magic numbers: During the initialization of Duqu, three decryption operations are performed, exactly as in Stuxnet. In case of Duqu, the compiled-in configuration is decrypted with a fixed decryption routine and it does not use any specific key, the variable configuration in the registry is decrypted with the key `0xAE240682` loaded from the compiled-in configuration, and the PNF file `netp191.pnf` is decrypted with the key `0xAE240682` loaded from the registry. The situation is the same for Stuxnet, the only difference is that the key loaded from the registry is different, and the decryption routines in Stuxnet are slightly different as well. In addition, in both cases, further configuration parameters are stored in a PNF file (in case of Duqu, this is `netp192.pnf` for the jminet7 variant and `cmi4464.pnf` for the cmi4432 variant), which starts with the magic number `0xAE790509`. The same magic is used in Stuxnet.

Injection targets: The injection target selection of Duqu and Stuxnet are very similar. Both Duqu and Stuxnet first check for known anti-virus products. Their checklists are essentially the same (even ordered in the same way), however, Rising Antivirus appears as an additional element in the list of Duqu. The injection target is then selected from a list of system processes including `svchost.exe`, `lsass.exe`, and `winlogon.exe`. The same list is used by Stuxnet. In addition, after injecting the malicious DLL payload in the target process, export _1 of the DLL is called in both cases.

Exported functions: The DLL in `netp191.pnf` contains 8 exports, while that in `cmi4432.pnf` has only 6 exports. In case of Stuxnet, the number of exports was 32; we suspect that the reason for this difference is the additional PLC functionality in Stuxnet, which is completely missing in Duqu. Nevertheless, the exports in Duqu show strong similarities to the non-PLC related exports in Stuxnet. For instance, exports _1 and _8 of `netp191.pnf` of Duqu are essentially the same as exports _1 and _32 of Stuxnet's `oam7a.pnf`. In both cases, these exports are related to RPC communications and they differ only in a few bits.

Import preparation by cheksums: Both Duqu and Stuxnet use the trick that some imports are prepared by looking up checksums in particular DLLs and comparing the results instead of directly naming the specific function to be called. The checksum calculation, however, seems to be different in Duqu and Stuxnet.

Hooks: The hook functions work in the exact same way in Duqu and in Stuxnet. In both cases, they use non-existent virtual files for libraries loaded from modules. Both Stuxnet and Duqu use the same 8 hooks in `ntdll.dll` during the injection process. Hooks used by rootkits are usually similar, however, the exact list of the hooks is specific to a given rootkit family and can serve as a fingerprint.

Communication module: Duqu has a backdoor covert channel control communication module that is used to send information to and receive commands from a remote Command and Control (C&C) center. In our case, the remote C&C server was located at the address 206.183.111.97, but later evidence shows that other instances used different servers. The communication protocol uses both HTTP port 80 and HTTPS port 443. The communication through port 443 is encrypted. The communication through port 80 starts with a valid HTTP request, followed by the transmission of (possibly encrypted) binary data obfuscated as jpeg images.

Keylogger module: Unlike Stuxnet, Duqu has a keylogger component that steals information from the infected system. The keylogger does not only log keystrokes, but it also regularly saves screenshots and packs other types of information. It stores data in the `%TEMP%` directory of the computer in a compressed format. The executable of the keylogger contains an embedded jpeg file. The jpeg image is not complete, the readable text shows "Interacting Galaxy System NGC 6745". This refers to a picture, taken from NASA, showing two colliding galaxies. Within the jpeg file, after the partial image, an encrypted DLL can be found which conatins the main keylogger functions.

2.1 Follow-up activities

Multiple security vendors pursued technical analysis of Duqu after our initial work. The most detailed results are from Symantec and Kaspersky. Their discoveries and conclusions are in-line with our results and observations and deepen the knowledge about the Duqu threat. Similarities to Stuxnet are common features of these analyses.

Multiple other Duqu infections were identified around the world, in a total number around 20, most of them in Europe and Middle East. We had no access to new samples, the information on this is based on the publicly available reports of the anti-virus industry. Based on these reports, we can say that Duqu is like a Lego-kit, and so is Stuxnet. Both of them are based on small components assembled together, they exist in several different versions with slight modifications, and they are created to perform their activity in a fast and efficient way. They are created to avoid identification using individual modifications and very careful error processing. This type of thinking about the threat was reinforced by Kaspersky in their recent report [2], where they reveal details about the discovery of previously unknown pieces of malware components related to both Duqu and Stuxnet. Consequently, Kaspersky introduced the name *Tilded platform* to refer to Stuxnet and Duqu as members of the same malware family. Although some still question that Duqu is made by the same authors as Stuxnet, each new evidence seems to support the hypothesis on their strong relation.

The zero-day exploit within the Duqu dropper was confirmed by Symantec and Microsoft in the last days of October 2011, and fixed by a patch in early December by Microsoft. Anti-virus vendors now include detections on parts

of the Duqu threat, and even generic detections on the exploit used by our known Duqu dropper, currently identified as CVE-2011-3402.

Speculations on the authors/origin of Duqu and Stuxnet are of high interest, but no sound evidence was found on this topic. There are still many questions unanswered on the threat and most likely, discussions, findings and follow-ups will be announced in the future.

3. DUQU DETECTOR TOOLKIT

Duqu is a sophisticated malware that has avoided detection for a period of time that shocked malware analysts. The exact start of the Duqu operation is still unsure today, but the stealthy time period of the malware spans several months, maybe years. The authors achieved this robustness with rigorous quality control, the use of advanced obfuscation techniques and thorough cleaning of activity traces.

Yet, thorough investigations uncovered several points where the malware authors could not fully cover their traces. We collected our observations and developed a set of heuristic tools to detect Duqu and, with a high chance, unknown variants of the Tilded platform. Given the potential impact of false negatives, our tools aim at completeness rather than precision and they require a careful investigation of results by security experts.

At the time of this writing, we provide six tools to heuristically detect Duqu variants and our tools can be broadly categorized into three areas: detecting file existence anomalies (*FindDuquSys*, *FindDuquTmp*, *FindPNFnoINF*), detecting properties of files and registry entries (*CalcPNFEntropy*, *FindDuquReg*) and analyzing code injection into running processes (*GetProcMem*). The output of these tools are stored in a log file, where suspicious files, memory regions, registry entries are indicated together with their corresponding hashes. Note that while some of these tools are rather simple and would be easy to defeat by changing the malware, they can still be used to detect existing infections. In addition, some of these tools are general and defeating them would require substantial change in the malware.

1. *FindDuquSys*. This tool recursively tries to find the loader executable component, the `.sys` kernel driver file of Duqu. It works similarly to signature based anti virus detectors and uses binary signature matching on all driver files in predefined directories, such as system32 drivers and System Volume Information directory[4]. The signature components were selected in a way that possibly modified versions of Duqu might be detected as well. It is not impossible, however, that our tool can detect these signatures in legitimate files, so if any string is detected, it is just an indication for the need of detailed manual analysis of the particular file. Care should be taken that running the program might need elevated privileges to successfully test all `.sys` files.

2. *FindDuquTmp*. The Duqu malware got its name after the usage of temporary files starting with ~DQ. In fact the detection tool seeks multiple types of temporary files used in Duqu:

   - The existence of `~DN1.tmp` shows that the keylogger/infostealer component might be installed on the computer. Our tool checks files recursively in predefined temporary directories, i.e., `Temp` directories of all the users and the Windows `Temp` directory.
   - `~DQ*` files might be related to the keylogger/infostealer log files. Some parts of the files are checked against Duqu's magics.
   - `~DF*` are compressed files created by a yet unknown part of Duqu and contain information gathered at the target computer. Our tool checks those files if they begin with a modified bzip magic, which shows that the temporary file is likely related to Duqu.

3. *FindPNFnoINF*. The PNF files installed by Duqu do not have the corresponding INF files. The tool checks all PNF files in Windows INF directory (located at `%WINDIR%\inf`), and indicates if some file does not have a related file with INF extension. Improper uninstallation of drivers can also cause such anomaly, so this does not necessarily signal the existence of Duqu. Experts should carefully check the results of the tool for false positives.

4. *CalcPNFEntropy*. This tool tries to find suspicious PNF files both in the Windows installation and System Volume Information directories. Both Duqu and Stuxnet put components in encrypted form into folder `%WINDIR%\inf` with a PNF extension. Encrypted and compressed files generally have a distinct characteristic: their entropy calculated over the binary file is larger then those of other standard binary files. This tool calculates entropy of all files in `%WINDIR%\inf`. Files with entropy above 0.6 are marked suspicious (calibrated by real-life Duqu samples with a typical entropy around 0.9).

5. *FindDuquReg*. This tool looks up the registry recursively from a given key node to identify suspicious entries with high entropy. In this regards, it works similarly to *CalcPNFEntropy*, but due to the very small size of binary data in registry entropy calculation is difficult. Therefore, instead of calculating entropy over bytes, we use four consecutive bits as one symbol using the same method as in *CalcPNFEntropy*.

6. *GetProcMem*. This tool builds upon the fact that Duqu injects itself into running processes such as `svchost.exe`, `lsass.exe` and creates a view of sections with read/write/execute rights. The technique used by the malware is a well-known code injection method, as it starts itself from a memory region that it previously wrote. The problem with detecting injection into running processes is that it may happen in case of benign software as well, therefore, this tool may generate false alarms. To limit the number of false positives, we only consider specific processes where Duqu typically injects itself.

We tested our toolkit on virtual machines infected by our Duqu sample and an available Stuxnet sample (Stuxnet.A). All the six tools in the toolkit generated alarms for Duqu

[4]This extension was introduced to be able to search deleted files as well.

infected machines. For the Stuxnet infected machines, naturally, the Duqu signature scanner and the temporary file detectors did not signal any problems, however the remaining four tools raised alarms. In all cases, we had a small number of false positive alarms, e.g., we found a few innocent PNF files without a corresponding INF file.

Our Duqu detector toolkit has been downloaded from more than 12000 distinct IP addresses distributed over 150 countries. The highest number of downloads originate from Vietnam, followed by the US, France, Iran, India, Poland, Norway, Hungary, Indonesia, and Great Britain.

4. DISCUSSION

Our investigation highlighted a number of interesting issues in the malware detection process and the general defense mechanisms against malicious software. Most of these issues relate to the trust we put into software components, the producers of software and the actors of the anti-malware industry.

4.1 Code signing

An intriguing feature of Duqu was that one of its kernel level components, the `cmi4432.sys` driver, had a valid digital signature that was generated with the private key of C-Media Electronic Inc., a company from Taiwan. The certificate of C-Media was issued by Verisign on August 3, 2009. It was a Class 3 certificate that provides the highest security level requiring, for instance, physical presence for authentication purposes in the certificate request phase. It is very likely that the attackers compromised the private key of C-Media and used the compromised key to generate the valid digital signature on the malicious driver. The certificate was revoked on October 14, as a result of sharing our Duqu analysis report with Symantec. In this case, the revocation process was very fast, probably because it was initiated by Symantec, the owner of Verisign.

Code signing is extensively used today to authenticate the identity of the producer of a software and the integrity of the code. A common assumption is that if code is signed then it can be trusted. As a consequence, many automated verification tools do not even check signed files, or they rely on the validity of signatures to filter false alarms.

However, a valid digital signature does not necessarily mean that the code is trustworthy. Technically, the validity of the signature only tells the verifier that the code has been signed by someone who possesses the private key, which does not exclude the possibility that the key is compromised. In addition, a valid signature does not tell anything about the trustworthiness of the signer, even if the key is intact.

These observations are not just theoretical arguments: signed programs with unwanted features or malicious intent do occur in practice [3]. In fact, there are multiple ways to get a piece of malware signed. First of all, attackers can setup a company, obtain a valid code signing certificate through regular procedures from any top level CA, and then sign their malicious software. Second, attackers can arrange the distribution of their software through a reseller that may put its own signature on it. There are examples for this sort of operation: for instance, the e-commerce outsourcing company Digital River signs binaries for its customers, and apparently some of those binaries also include rogue and malicious code [3]. Third, attackers can steal the code signing key of an established company either by breaking in the company's IT system or simply by bribing an employee. This is facilitated by storing code signing keys on developer machines that are connected to the Internet, and not using password protected storage for those keys or storing the password in a batch file for convenience; these are practices that are more common than they should be. Finally, attackers can break into developer machines and place their malware in the software that is being developed by the company before it is signed. Most malicious programs with a valid digital signature belong to the first category, but Duqu and Stuxnet illustrate that code signing keys can also be compromised in practice.

The problem that we face is that the current infrastructure and practices used for code signing cannot address this issue effectively. While some CAs have strict authentication policies when evaluating a certificate request, we are not aware of any periodic audits after the issuing of the certificate aiming at the verification of how the private keys are handled and used by the certificate owner. Similarly, we have not heard about any case when the certificate of a software maker was revoked due to its negligence in the key management and code signing procedures. Therefore, software companies have no real incentives to follow strict key management policies, while there is a temptation for neglecting even the basic precautions for the sake of efficiency and convenience. Another problem is that it is not clear who actually should perform the auditing of software companies. Letting the CAs perform the audits would not be scalable, and it would be too costly for them. In addition, a CA can revoke the certificates of a company if it is detected negligent, but it cannot carry out any further actions against the company, which can then continue its operation and try to find another CA that would issue certificates for its code signing keys. So the CA has no incentives to do strict verifications, because it can lose clients and profit, while less diligent CAs may prosper in the market.

To summarize, while we believe that, in principle, code signing is a useful feature as a first line of defense, because it raises the barrier for attackers, we emphasize that one should not fully trust code even if it is signed, and we argue that the practice of code signing today is far from being satisfactory: there are misplaced incentives and scalability problems, leading to negligent key management, which then limits the effectiveness of code signing as a mechanism to establish trust in software. Finally, we note that these problems need urgent solutions, because the attackers' demand for being able to sign their malware is expected to grow rapidly in the future due to the fact that unsigned software can no longer be installed on recent and future versions of Windows without warning messages, if at all.

4.2 Signature based detection

Signature based malware detection is important, as it is the most effective way of detecting known malware, however, Duqu and other recent targeted attacks clearly show that it is not sufficient. In fact, the creators of high-profile targeted threats such as Duqu and Stuxnet have the resources to fine-tune their malware until it passes the verification of all known anti-virus products, therefore, such threats will basically never be detected by signature based tools before they are identified and their signatures are added to the signature database.

A solution could be heuristic anomaly detection. Indeed,

some anti-virus vendors have already started to extend their signature based tools with heuristic solutions. An interesting approach seems to be binary reputation heuristics (e.g., detecting "rare" binaries). While these techniques are far from being sufficiently reliable to effectively prevent targeted attacks, they are certainly a first step away from standard signature based detection. The basic problem with anomaly detection based tools is that they generate false alarms, and it is difficult to filter those false positives in an automated manner. However, more work on white listing techniques and cloud based information sharing may improve the situation. Academic research could contribute a lot in this area, because the problems require highly innovative solutions.

We should also mention that, in some application areas, false positives may be tolerated better, because missing a real attack has devastating consequences. In particular, we believe that in the domain of critical infrastructures, such as nuclear power plants, chemical factories, certain transportation systems, and so on, where a successful logical attack on the connected IT infrastructure may lead to a fatal physical accident, false positives should be tolerated, and there should be expert personnel available to handle them. This observation motivated our work on the Duqu detector toolkit. While that toolkit is somewhat customized for detecting Duqu variants, in the future, we expect more general tools based on similar principles.

4.3 Information sharing

Once a high-profile incident like the Duqu malware infection is detected, the most important tasks are to: (a) contain and mitigate the incident locally and (b) to disseminate the intelligence information to mitigate the global effects of the malware. When we were asked to handle the intrusion by the Duqu malware at the European firm, we quickly observed the anomalies and did a thorough investigation to contain the malware locally. As the investigations unfolded, we discovered relation to Stuxnet and realized that the threat is global. Thus we advocated a security response that involved contacting reputable security vendors for global information dissemination and threat mitigation.

The biggest challenge in threat mitigation was to establish trust between the client firm and selected security vendors for information sharing. Although there is an established industry standard among security vendors for sharing information there is an important and seemingly unbridgeable gap between the end users and security vendors. Clearly, the firm's driving incentive was to decrease the impact of the incidents at their own premises. We emerged as *mediators* between these parties and convinced the firm to enter the information and sample sharing process with the most reliable security vendors. Anecdotal evidence suggests that, unlike in our case, security vendors are often unable to obtain forensics information even when their product detected infection. Given the importance of targeted attacks, we believe that increasing trust to enable forensics is a key enabler of global incident response.

When we discovered the dropper in an MS Word document obtained from our client, we immediately realized that this critical information needs to be shared with security vendors to aid their detection efficiency. Nonetheless there was a key issue with the dropper file: it contained sensitive information about the client and we did not have the expertise to sanitize the file such that it becomes appropriate for sharing. Given the trust that the company had in us, and the trust accumulated between Symantec and us during the handling of the incident, we took advantage of our role as a mediator, and we convinced our client to let Symantec produce an anonymized dropper for sharing. This dropper sample was critical in proving the existence of the zero-day exploit and motivate Microsoft to handle the case with high priority. To the best of our knowledge, only one other dropper has been discovered since.

Clearly, the solution we followed does not scale. According to our experience, very few end-user firms are able to produce sanitized security information that can be shared for global incident response. While security vendors possess the knowledge to produce sanitized samples, the process is demanding and highly personalized at the moment. This implies again that trust needs to be established between end-users and security experts who are able to prepare forensics evidence and protecting the firm's identity at the same time. Furthermore, to ease the load on these experts, we need to seek semi-automatic production of anonymized forensics evidence as a key challenge.

5. CONCLUSIONS

The main message of this paper is that the white hat community should catch up on all fronts: (i) code signing as a preventive measure has limitations and the current practice raises questions concerning the effectiveness of code signing as a means to establish trust in software, (ii) more research should be devoted to heuristic anomaly detection tools to complement signature based scanners, and (iii) new concepts and mechanisms are needed that help information sharing between victims of incidents and the anti-virus industry to help incident handling and forensic analysis.

6. ACKNOWLEDGMENTS

We are thankful to Eric Chien and his colleagues at Symantec, and especially, in the Security Response team for the wonderful collaboration at various stages of the Duqu analysis project and for the sanitization of the dropper file. We are thankful to Alex Gostev and Costin Raiu from Kaspersky for the useful discussions and for their positive feedback on our Duqu detector toolkit. We are also thankful to Peter Szor from McAfee for his support. Finally, we are thankful to our client for their positive attitude towards the research on Duqu.

7. REFERENCES

[1] B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi. Duqu: A Stuxnet-like malware found in the wild. Technical Report Version 0.93, CrySyS Lab, BME, October 14 2011.

[2] A. Gostev and I. Soumenkov. Stuxnet/Duqu: The evolution of drivers. Technical report, Kaspersky, December 28 2011.

[3] J. Niemelä. It's Signed, therefore it's Clean, right? Presentation at CARO 2010, available at `http://www.f-secure.com/weblog/archives/Jarno_Niemela_its_signed.pdf`, 2010.

[4] Symantec Security Response. W32.Duqu: The precursor to the next Stuxnet. Technical Report Version 1.0, Symantec, October 18 2011.